TÉCNICO LISBOA

Multi-UAV Mission Coordination using Signal Temporal Logic Specifications

Nuno Ricardo Ferreira Duarte

Thesis to obtain the Master of Science Degree in

Electrical and Computers Engineering

Supervisor(s): Prof. Pedro Manuel Urbano de Almeida Lima
Dr. Richard Murray

Examination Committee

Chairperson: Prof. João Fernando Cardoso Silva Sequeira
Supervisor: Prof. Pedro Manuel Urbano de Almeida Lima
Member of the Committee: Prof. Rita Maria Mendes de Almeida Correia da Cunha

September 2016

Dedicated to my family.

Acknowledgments

As I am writing these last words to finish my thesis, the realization that this document will be the very last milestone for graduating from my study at Instituto Superior Técnico, is slowly sicking in. It has been an exciting, sometimes difficult, but always very interesting journey, and my thesis project was no different.

First of all I would like to thank my family and friends, for not only supporting me during the complete time of my study in IST, but also for being understanding whenever I was stressed out or easily irritable, especially during the final weeks of my thesis project.

Then to my two supervisors Prof. Pedro Lima and Prof. Richard Murray whose support was instrumental for this thesis and who was available to me whenever I needed.

I want to give my special thanks to Prof. Richard Murray and Samira Farahani for giving me the opportunity to work on a summer internship in Caltech. Without it I would never have learned about Signal Temporal Logic and I wouldn't have done a master thesis on the subject.

I would also like to thank all my friends who supported me and made me feel happy especially during the difficult times.

Resumo

Lógica temporal tem sido usado com sucesso numa grande variedade de sistemas e ambientes. Anteriormente, trabalhos realizados usando Lógica temporal linear (LTL) desenvolveram controladores especificando propriedades de sinais em tempo discreto. No entanto, estes mesmos controladores não são os mais apropriados para lidar com sinais continuos com dependência na variável de tempo, mais especificamente, a optimizar trajetórias em espaços com perturbâncias, obstáculos estáticos e dinâmicos, conhecidos à priori ou não. Para resolver estes problemas, nós usamos Lógica temporal de sinais (STL) com o intuito de desenvolver um controlador para uma missão de busca e salvamento num ambiente totalmente conhecido; a missão envolve mais do que um veículo aéreo não tripulado (UAV) a trabalhar cooperativamente para satisfazer certas tarefas tais como, monitorizar areas, planeamento de trajetórias, satisfazer a seguranca de cada um, e desviar de obstáculos que tanto podem ser obstáculos presentes no ambiente como outros veículos presentes na missão. Para desviar de obstáculos foi desenvolvido vários sistemas, um dos quais, envolvendo especificações STL que têm em conta a localização de todos os veículos.

As simulações provam que este controlador desenvolvido consegue gerar trajectórias para uma missão que envolve 3 UAVs que cooperativamente vão de encontro ao objecto, e de seguida procedem ao seu transporte. Para além disso, foi obtido com sucesso um controlador usando STL para especificar uma missão em que 2 UAVs sincronizam o seu comportamento de modo a transportar um objecto de grandes dimensões. Esta missão ocorreu sem qualquer colisão durante o trajecto.

Palavras-chave: Lógica, Planeamento, UAV, STL.

Abstract

Temporal Logic has successfully been used for specifying a wide range of behaviors for systems and environments. Previous works using Linear Temporal Logic (LTL) have synthesized controllers when specifying properties of discrete time signals. However, these controllers are inconvenient to handle continuous signals with time dependency, more specifically, when optimizing trajectories under environmental uncertainties, such as static or dynamical obstacles, known and unknown obstacles. To address these issues, we use Signal Temporal Logic (STL) instead and our aim is to design a reactive strategy and trajectory planning for a search and rescue mission in a known environment; the mission involves more than one autonomous aerial vehicle (UAV) working together to satisfy tasks such as periodically monitoring areas, path planning, staying safe, and avoiding obstacles that can either be object in the environment or a different autonomous vehicle that is attempting to satisfy its own tasks. To avoid obstacles we will compare more that one solution using different methods such as STL specifications that take into account the localization of all vehicles in the environment or a centralized node that checks the localization of two vehicles in order to prevent any possible future collisions.

The simulations show that using a controller developed using this approach can generate trajectories for a mission involving 3 UAVs performing a cooperative mission of grabbing 3 individual objects and 1 object that requires 2 UAVs for carrying with successful results and without any collisions.

Keywords: Logic, Planning, UAV, STL.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, state-of-the-art control and navigation algorithms - such as motion planning, reach avoid tasks, reference tracking, and stability - have been applied to the extended application domains for these vehicles. Research areas include both military and civilian applications (such as intelligence, reconnaissance, surveillance, exploration of dangerous environments) where robots and UAVs can replace humans.

1.1 International Competition

The mention case study for this thesis came from an international competition, the Mohamed Bin Zayed International Robotics Challenge (MBZIRC) happening in 2017 in Abu Dhabi (http://www.mbzirc.com/) and is held every two years. This competition invests on new advances in the field of robotics promoting awareness to the broader public of the tremendous value of this field.

MBZIRC 2017 will consist of three challenges and a triathlon type Grand Challenge:

1. **Challenge 1** requires UAV to locate, track and land on a moving ground vehicle.
2. **Challenge 2** requires an unmanned ground vehicle (UGV) to locate and reach a panel, and physically operate a valve on the panel.
3. **Challenge 3** requires a team of UAVs to collaborate to search for, locate, track, pick up, and place down a set of static and moving objects.
4. **The Grand Challenge** requires a team of robots (UAVs and UGVs) to compete in a triathlon type event that combines Challenges 1, 2 and 3.

A UGV is a vehicle that operates while in contact with the ground, can also be known as autonomous automobile or a robotic car.

Three independent challenges complete this international event in which every team who participates can choose from which challenge they would like to participate or they choose to tackle all the challenges and try to solve the three challenges in the same turn. In our case, the focus is on **Challenge 3**. This challenge, also known as the ”Treasure Hunt” consist of a team of 3 UAVs that search, find, pick and relocate a number of static and/or moving objects. The initial challenge is to collect and relocate objects in a known environment with a team of 3 quadrotors. In the matter of this thesis the process of grabbing the objects by the quadrotors is not relevant since that is not the goal of this thesis nor is it a challenge that will have any advantage to be solved with logic formulation. The reason for us to have chosen this challenge as a base line to our problem is because of the complexity of the mission. Involving more than one vehicle to find more than one goal requires coordination and efficiency, two things that may provide tremendous feedback into how complex and feasible a controller using STL specs may be.

Figure 1.1: Challenge 3 of the International Competition

In addition of each quadrotor grabbing an object from the environment and relocate to a specific goal, there are also large size objects that must be place to the same location. This requires synchronization of the quadrotors to move the large object safely and as stable as possible which will be a very interesting task to solve using logic operations and logic components. Even though this challenge is to be played at an outdoor environment and the location of the objects won’t be known a priori, for the purpose of this thesis those details are not taken into account for the solution of the problem. as they concern other functions of the quadrotor that are not our responsibility to tackle. Its the specification of the coordinated mission and the generation of a feasible trajectory that we are going to approach.

1.2 A Brief History of UAV’s

Unmanned aerial vehicle or simply UAV is, as the name suggests, an aircraft without any onboard pilot controlling its trajectory. Some are used for commercial use, e.g. toys, others for military use as aerial strikes or surveillance. For the purpose of this thesis we are going to to focus on the former which has been of great interest to the academic world. An autonomous aerial vehicle can take many forms depending on the purpose of the experiment and also the amount of rotors it has. Many research breakthroughs have been achieved for different types of autonomous UAVs, such as helicopters, quadrotor, hexarotor, etc. In this thesis our work is going to be developed for the purpose of being applied to quadrotors, more commonly referred as a quadcopter however, and to keep a concise nomenclature, we will name a vehicle with 4 rotors as a quadrotor. We can also see a representation of a quadrotor in Figure 1.2.

Figure 1.2: An example of a quadrotor

1.3 Motivation

The focus of this thesis is on multi-UAV mission planning using signal temporal logic (STL) specifications. Mission coordination has been studied by many researchers and in various problem settings [3], [32], [30]. It is clear advantageous that it is to develop a coordinated team of unmanned vehicles to achieve a goal instead of using a single vehicle. Additionally, missions involving surveillance, rescue and reaching goals have been increasing in complexity. In this context, it is important to develop new aways of describing the missions specifications. STL is a formal language used to define constraints and specifications of continuous signals. The use of this language instead of other well-known languages ([19], [11] and [18]), is due to its application in continuous and hybrid systems. Combining a formal language to specify the tasks of multiple autonomous vehicles will provide tremendous insight to the performance of such specifications in a real-life applications.

1.4 Objectives

The Objectives of this thesis are as follows. First, is to implement a controller capable of generating a path for a flee of three UAVs to plan a mission to collect three separate objects in a dynamic environment and collect each object to a specific location. This will be achieved using STL specifications as MIL constraints which will be introduced in the coming chapters. To showcase the solution obtained by the controller, a simulator running in Gazebo will be used which incorporates the UAV dynamics as well as the characteristics of the environment and objects. In the second part of thesis it is proposed a method for cooperative planning of UAVs to grasp an object with over-size proportions, and its safe transportation to a specific location.

1.5 Thesis Outline

In Chapter 2, the methods of STL are introduced and mixed integer linear programming formulations are provided. In the same chapter, a view on mission coordination and a comparison with previous temporal logic languages are introduced. Chapter 3 will begin by analyzing the quadrotor physical process. Later on we will discuss the controller implemented to send control inputs to the quadrotor model. In Chapters 4, an example of a MILP encoding is represent to address constraints specified in STL. Chapter 5 will show the experimental results obtained by the feasible controller for every respective task. And finally Chapter 6 will discuss several key observations, and point to the directions for our future work.

Chapter 2

Background

This chapter introduces the preliminaries used in the thesis. We provide definition of signal temporal logic (STL) including a comprehensive study of temporal operators. More background information and details of the methodology needed for this thesis are discussed. A brief introduction to Linear Temporal Logic (LTL) and its semantics are given as it is essential to the understanding of the STL formulation.

2.1 Mission Coordination

Mission coordination has been discussed for several years, either for humanitarian missions or with a more military view [3], [32], [30]. Nonetheless, using aerial vehicles in a group instead of a single quadrotor is more advantageous in terms of allowing solutions for more complex tasks and also being economically reliable than using a single UAV with larger proportions and higher quality sensors and actuators. Mission coordination involving multi-UAV has been a topic of tremendous research using several different perspectives, and describing formal specifications with mixed-integer linear constraints of discrete or continuous signals has been one of the most popular. Logic languages are the most intuitive and easy to understand even though it can be very computationally demanding for complex problems. Temporal Logic such as Linear Temporal Logic (LTL), Metric Temporal Logic (MTL), or Computational Tree Logic (CTL) are some of the most well used to describe specifications for path planning. In this thesis, some background to the work done with some of those languages will be presented and comparisons to the alternative chosen for this work.

2.2 Linear Temporal Logic

Linear Temporal Logic (LTL) has been introduced as a language that formalizes specifications of dynamical control systems [10], [15], [14], [21]. It is able to achieve that by checking the satisfaction of these specifications for a given control system, and with that, design a control systems that can satisfy the given specifications. Using temporal logic as a specification, allows us to develop model-checking techniques [8] in order to decide whether a given problem is satisfied by a set of temporal specifications.

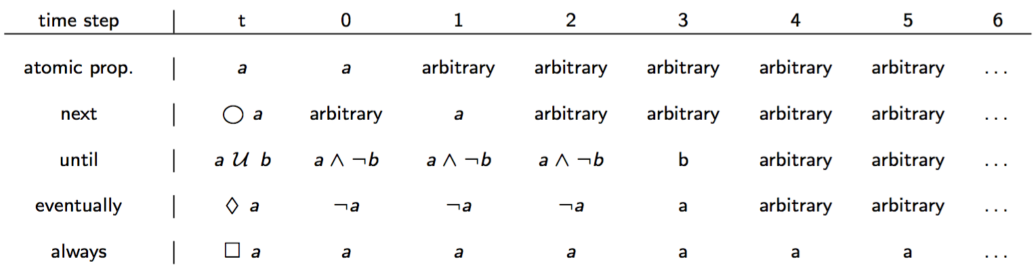
Temporal logic has been used since the beginning of the 80s for planning in Artificial Intelligence [12] as well as in Control Theory [13].

The following syntax are in accordance with the handbook of Principles in Model Checking section 5 [2]. The basic ingredients of LTL-formulae are *Atomic Propositions* (*AP*), the Boolean connectors like conjunction $\wedge$, disjunction $\vee$, and negation $\neg$, and two basic temporal modalities $\bigcirc$ (pronounced ”next”) and $\mathcal{U}$ (pronounced ”until). LTL is built from a set of *Atomic Propositions*. Those AP are formed according to the following grammar:

$$\varphi ::= \text{true}(\top) \mid a \mid \neg\varphi \mid \varphi \vee \psi \mid \bigcirc\, \varphi \mid \varphi\, \mathcal{U}\, \psi$$

where $a \in$ *AP*. The standard Boolean operators $\Rightarrow$ (implies) and $\Leftrightarrow$ (equivalent) can be defined as $p \Rightarrow q = \neg p \vee q$ and $p \Leftrightarrow q = (p \Rightarrow q) \wedge (q \Rightarrow p)$, respectively. Abbreviations for LTL formulas are derived temporal operators $\Diamond\varphi = True\, \mathcal{U}\, \varphi$ (eventually), $\Box\varphi = \neg\, (\Diamond\, \neg\, \varphi)$ (always). This temporal operators are going to be applied in the next section for STL formulation with the addition of time specifications.

Figure 2.1 sketches the intuitive meaning of temporal modalities for the simple case in which the arguments of the modalities are just atomic propositions from $\{a, b\}$. On the left-hand side, some LTL formulae are indicated, whereas on the right hand side sequences of states (i.e., paths) are depicted.



time step	t	0	1	2	3	4	5	6
atomic prop.	a	a	arbitrary	arbitrary	arbitrary	arbitrary	arbitrary	...
next	$\bigcirc\ a$	arbitrary	a	arbitrary	arbitrary	arbitrary	arbitrary	...
until	$a\ \mathcal{U}\ b$	$a \wedge \neg b$	$a \wedge \neg b$	$a \wedge \neg b$	b	arbitrary	arbitrary	...
eventually	$\Diamond\ a$	$\neg a$	$\neg a$	$\neg a$	a	arbitrary	arbitrary	...
always	$\Box\ a$	a	a	a	a	a	a	...

Figure 2.1: Intuitive semantics of temporal modalities.

By combining the temporal modalities $\Diamond$ and $\Box$, new temporal modalities are obtained. For instance, $\Box\Diamond\varphi$ (”always eventually φ”) described the (path) property stating that at any moment j there is a moment $i \geq j$ at which an φ-state is visited. This thus amounts to assert that an φ-state is visited infinitely often. On the other hand, $\Diamond\Box\varphi$ (”eventually always” φ) expresses that from some moment j on, only φ-states are visited.

2.3 Metric Temporal Logic

When adapting LTL for real time, see e.g. [20], [26], [27], two observation should be heeded. First, unlike discrete state sequences, the time domain of expected behavior is dense. Therefore, the semantics of **X** cannot be clearly designated, as for given time point, there is no ”next” state. The temporal operator itself may be then omitted, despite LTL without $\mathcal{X}$ losing expressiveness (it is unable to discriminate ”stuttering” sequences [8]).

Second, when introducing timing constraints (which is part of the motivation for MTL introduction), they form a complement for temporal operators, as other operators do not reason about time.

Hence, the syntax of MTL is defined as follows:

$$\varphi ::= \text{true}(\top) \mid a \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi\, \mathcal{U}_{\mathcal{I}}\, \psi$$

where I is an interval on $\mathbb{R}_0^+$ (i.e. the time domain). Syntactic shortcuts in the form of the other Boolean operators and temporal operators $\mathcal{F}_I$ (”eventually”) and $\mathcal{G}_I$ (”always”) can be also included.

MTL possess both discrete and continuous semantics, as it can be interpreted over both infinite timed state sequences and signals. Similarly to LTL, predicates specify subsets of X and satisfaction of formula is given for each time point of a signal or each state of the timed state sequence. However, operator $\mathcal{U}$ is associated with an interval which describes when the release condition ψ must become valid for the formula to be satisfied. For example, the formula $\varphi\, \mathcal{U}_{[3,4]}\, \psi$ means that φ will be valid until ψ is valid at some $t \in [3, 4]$. Obviously, unconstrained until corresponds to $\mathcal{U}_{[0,\infty)}$. The same applies to the other temporal operators with bounded intervals.

$\Diamond_{[3,4]}\varphi$ φ will become true within 3 to 4 time units from now.

$\Box_{[3,4]}\varphi$ φ will be valid between time t = 3 time units to time t = 4 time units.

$\Box_{[0,\infty)}\Diamond_{[3,4]}\varphi$ φ will be true at a time $t \in$ [3, 4] and this will happen infinitely often

$\Diamond_{[3,4]}\Box_{[0,\infty)}\varphi$ after φ is true between the time interval $t \in$ [3, 4], φ will be true from then on.

The last two examples are the combination of the temporal operators $\Box$ and $\Diamond$ corresponding to two distinguished properties of logic specification. With similar characteristics as the ones mentioned in LTL, $\Box\Diamond$ means ”always eventually”, and so in order to reproduce this periodic aspect over the whole period time, the time interval is represented as the total duration $\Box_{[0,\infty)}$. The latter example which corresponds to $\Diamond\Box$ meaning ”eventually always”, with the purpose of representing a convergence to some action after some period of time, is also defined as $\Box_{[0,\infty)}$ giving the action to be true after some time between $t \in$ [3, 4] and remaining true for the rest of the duration of the simulation. These are the possible alternatives of representation of the different specifications that this thesis is going to use and any other possible combination is not regarded as important or useful in our case.

2.3.1 Multi-UAV planning using TL languages

Research about multi-UAV mission planning using temporal logic (TL) languages has been done in recent years, see for e.g in LTL [19], [35], [34], and in MTL [11], [18]. In the different papers, it was shown that interesting multi-UAV mission planning scenarios can be modeled naturally using LTL, while optimal plans for problems of practical sizes can be computed in reasonable amount of time. Even though only multiple-UAV mission planning applications have been considered, it should be noted that several other logistics problems with complex timing constraints can be modeled using the temporal logic specifications. As for MTL specifications, it is emphasized the significance of relative timing and constraints for Multiple-UAV mission planning problems. This reinforces the need and also the importance of using temporal logic specifications based on MILP computational method.

2.4 Signal Temporal Logic

Most of the notation in this section mirrors that in [28] and [29], and missing details can be found in either works.

2.4.1 Discrete-Time Systems

We consider discrete-time systems of the form

$$x_{t+1} = f(x_t, u_t, w_t) \tag{2.1}$$

where $t = 0, 1, \dots$ indicate the time steps, $x_t \in \mathcal{X} \subseteq (\mathbb{R}^{n_c} \times \{0,1\}^{n_l}$ denotes the states, $u_T \in U \subseteq (\mathbb{R}^{m_c} \times \{0,1\}^{m_l})$ denotes the control inputs, $w_t \in W \subseteq (\mathbb{R}^{e_c} \times \{0,1\}^{e_l}$ denotes the external inputs or disturbances, and the initial state is $x_0 \in \mathcal{X}$. Note that all the variables may have real-valued or binary components. A *run* $\sigma = (x_0 u_0 w_0)(x_1 u_1 w_1)(x_2 u_2 w_2)\dots$ is an infinite sequence of state, input, and disturbance components at each time step t. Similar to [28], the horizon-*N* run of a system modeled by (1), denoted by $\sigma = (x_0, \mathbf{u}^N, \mathbf{w}^N) = (x_{0,0}, w_0)(x_1, u_1, w_1)(x_2, u_2, w_2)\dots(x_N, u_N, w_N)$ is assumed to be unique for an initial state $x_0 \in X$, a control input sequence $\mathbf{u}^N = u_0 u_1 u_2 \dots u_{N-1} \in U^N$ and a sequence of environment inputs $\mathbf{w}^N = w_0 w_1 w_2 \dots w_{N-1} \in W^N$. We also consider a generic cost function $J(\sigma(x_0, \mathbf{u}.\mathbf{w}))$ that maps (infinite and finite) runs to $\mathbb{R}$.

2.4.2 Syntax

STL formulas are defined recursively according to the grammar

$$\varphi ::= \mu \mid \neg\mu \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \square_{[a,b]}\, \psi \mid \varphi\, \mathcal{U}_{[a,b]}\, \psi$$

where μ is a predicate whose value is determined by the sign of a function of an underlying signal **x**, i.e., $\mu \equiv \mu(x) > 0$, and ψ is an STL formula. The predicate $\neg\mu$ is the negation of the predicate μ, using boolean operators such as conjunction ($\wedge$), disjunction ($\vee$), or until ($\mathcal{U}$) we can write a set of predicates denominated as a STL formula ψ. From [2] temporal operators $\Diamond$ (eventually) and $\Box$ (always) can be derived from the previous boolean operators.

All the STL formulas can be defined with respect to signal **x** at time *t* in the following manner

$$
\begin{aligned}
(\mathbf{x}, t) \models \mu &\Leftrightarrow \mu(x_t) > 0 \\
(\mathbf{x}, t) \models \neg\mu &\Leftrightarrow \neg((\mathbf{x}, t) \models \mu) \\
(\mathbf{x}, t) \models \varphi \wedge \psi &\Leftrightarrow (\mathbf{x}, t) \models \varphi \wedge (\mathbf{x}, t) \models \psi \\
(\mathbf{x}, t) \models \varphi \vee \psi &\Leftrightarrow (\mathbf{x}, t) \models \varphi \vee (\mathbf{x}, t) \models \psi \\
(\mathbf{x}, t) \models \Box_{[a,b]}\varphi &\Leftrightarrow \forall t' \in [t+a, t+b], (\mathbf{x}, t') \models \varphi \\
(\mathbf{x}, t) \models \varphi\, \mathcal{U}_{[a,b]}\ \psi &\Leftrightarrow \exists t' \in [t+a, t+b] \text{ s.t. } (\mathbf{x}, t') \models \psi \\
&\quad \wedge \forall t'' \in [t, t'], (\mathbf{x}, t') \models \varphi
\end{aligned}
$$

A signal $\mathbf{x} = x_0x_1x_2\ldots$ satisfies φ, denoted by $\mathbf{x} \models \varphi$, if $(\mathbf{x}, 0) \models \varphi$. Informally, $\mathbf{x} \models \Box_{[a,b]}\varphi$ if φ holds at every time step before *a* and *b*, and $\mathbf{x} \models \varphi\, \mathcal{U}_{[a,b]}\ \psi$ if φ holds at some time step between *a* and *b* until ψ becomes true. Additionally, $\Diamond_{[a,b]}\varphi = \top\, \mathcal{U}_{[a,b]}\ \varphi$, so that $\mathbf{x} \models \Diamond_{[a,b]}\varphi$ if φ holds at some time step between a and b. STL formulas refer to discrete time intervals of discrete-time systems.

An STL formula φ is *bounded-time* if it contains no unbounded operators; the $bound$ of φ is the maximum over the sums of all nested upper bounds on the temporal operators, and provides a conservative maximum trajectory length required to decided its satisfiability. For example, for $\Box_{[0,4]}\Diamond_{[3,6]}\varphi$, a trajectory of length $N \geq 5 + 4 = 9$ is sufficient to determine whether the formula is satisfiable.

2.4.3 Robust Satisfaction of STL formulas and MILP encoding

In this work, we consider the quantitative or robust semantics for STL, which assigns a real value to a predicate as an indication of to what extend a formula is satisfied. Following [28], the robustness function is defined as a real-valued function ρ^{φ} of signal **x** and t such that $\rho^{\varphi}(\mathbf{x}, t) > 0 \equiv (\mathbf{x}, t) \models \varphi$ and they are computed recursively as follows

$$
\begin{aligned}
\rho^{\mu}(\mathbf{x}, t) &= \mu(x_t) \\
\rho^{\neg\mu}(\mathbf{x}, t) &= -\mu(x_t) \\
\rho^{\varphi\wedge\psi}(\mathbf{x}, t) &= \min(\rho^{\varphi}(\mathbf{x}, t), \rho^{\psi}(\mathbf{x}, t)) \\
\rho^{\varphi\vee\psi}(\mathbf{x}, t) &= \max(\rho^{\varphi}(\mathbf{x}, t), \rho^{\psi}(\mathbf{x}, t)) \\
\rho^{\Box_{[a,b]}\varphi}(\mathbf{x}, t) &= \min_{t' \in [t+a, t+b]} \rho^{\varphi}(\mathbf{x}, t') \\
\rho^{\varphi\ \mathcal{U}_{[a,b]}\ \psi}(\mathbf{x}, t) &= \max_{t' \in [t+a, t+b]} (\min(\rho^{\psi}(\mathbf{x}, t'), \\
&\qquad \min_{t'' \in [t, t'] \rho^{\varphi}(\mathbf{x}, t''))}
\end{aligned}
$$

For example, the robust satisfaction of $\mu_1 \equiv x - 5 > 0$ at time 0 is $\rho^{\mu_1}(x, 0) = x_0 - 5$.

To avoid the curse of dimensionality of the state space abstraction, STL specifications can be encoded as mixed-integer linear constraints [28]. Accordingly, we restrict ourselves to discrete time systems and to linear or affine predicates. Note that in case of dealing with continuous dynamics, we can approximate them by their equivalent discretized model using an appropriate sampling time. Using this encoding, the optimal input sequence for system 2.1 (as will be shown in the next section) can be obtained by solving a mixed integer linear programming (MILP) optimization problem at each time step. The interested reader is referred to [28] for the details of this encoding.

The advantage of this *robustness-based* encoding is that it allows us to obtain a trajectory that maximizes or minimizes the robustness of satisfaction. The objective function for the optimization problem can be then defined as the robustness function, which will be maximized, or the robustness function can be a complement to domain-specific objectives on runs of the system.

As mentioned in [28], MILPs are NP-hard, and hence impractical when the dimensions of the problem grow. We present the computational costs of the above encoding in terms of the number of variables and constraints in the resulting MILP. If *P* is the set of predicates used in the formula and $|\varphi|$ is the length (i.e. the number of operators), and N is the number of STL formulas, then $O(N\dot{|}P|) + O(N\dot{|}\psi|)$ continuous variables are introduced. In addition, $O(N)$ binary variables are introduced for every instance of a Boolean operator, i.e. $O(N\dot{|}\varphi|)$ Boolean variables.

2.5 Comparing different Temporal Logics

To understand the need of using STL specifications in comparison to previous languages and also what motivated this contribution a simple yet concise explanation of the differences between each ramification of the temporal logic formulation is provided here. The simple example means that whenever signal r is true it implies that eventually signal g will be true in a given time. This of course will be expressed differently depending on the formulation used and in the case of real-time languages the temporal operators have a time interval.

$$\text{LTL} \quad \Box(r \Rightarrow \Diamond g)$$

$$\text{MTL} \quad \Box(r \Rightarrow \Diamond_{[0,0.5s]} g)$$

$$\text{STL} \quad \Box(x[t] > 0 \Rightarrow \Diamond_{[0,0.5s]} y[t] > 0)$$

MTL and STL derived from LTL as extensions with real-time and real-valued constraints [36]. For LTL specifications the signals are boolean predicates in discrete-time. In MTL the signals are also represented as boolean predicates however it is now possible to specify in real-time. Finally, using STL specifications is now possible to describe the signal as predicates over real values and in real-time. This in return allows for better description of the mission since the variables that are being manipulated are the degrees of freedom from the UAV model.

Chapter 3

Quadcopter Model, Control and Simulation

In order to achieve the desired objectives it is necessary to acquire some basics of the dynamics of a quadcopter. In this chapter, it is presented a bottom-up approach. This starts by defining the coordinate systems, as well as the broad concepts of the vehicle, the mathematical description and the control method used. In this section a short explanation of the Gazebo simulation and all its components required for the control and movement of the quadrotor is presented [31]. To finish with this chapter a PI controller is developed in MATLAB to simulate the controller implemented in Gazebo for the velocity commands.

3.1 Definitions and Basic Concepts

The quadrotor model is a system with 6 DOF described with twelve states, six of them corresponding to the attitude and the other six states are the positions with respect to the world or to a specific location in the environment (object, goal, etc). For the six states the variables ϕ, θ and ψ (roll, pitch, and yaw angles are Euler angles between the body-fixed frame and the world-fixed frame) and $p, q,$ and r $(rad.s^{-1})$(roll rate, pitch rate and yaw rate with respect to the body-fixed frame) are determined using the right-hand rule. The remaining six states of the quadrotor are x, y, and z which denote the position of the center of mass of the vehicle.

In order to move the quadrotor the linear and angular speed of the rotors needs to change. Increasing or decreasing the speed of all 4 rotors will cause the quadrotor to increase or decrease its height, respectively. Moreover, changing the speed of pair rotors (1,3) or (2,4) will originate in the rotation of the quadrotor around the z axis (yaw rate). The pitch rate, a rotation around the z axis, is achieved when there is a difference of speed between pair rotors (1,3). If we apply the same principle to the pair rotors (2, 4) we get the third angular velocity known as roll rate.

The earth-fixed inertial reference frame is $E_1(e_{11}, e_{21}, e_{31})$ and the body-fixed reference frame is $E_B(e_{1B}, e_{2B}, e_{3B})$. The absolute position of the quadrotor is described by **X** $= [x, y, z]^T$ (m) and its

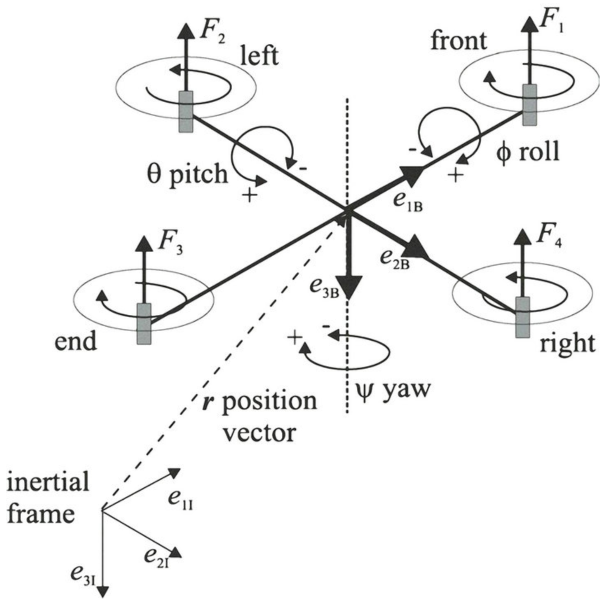


Figure 3.1: from Attitude Control of a Quadrotor with Optimized PID Controller, Bolandi

attitude by the Euler angles $\Theta = [\phi, \theta, \psi]^T \ (rad)$, used corresponding to aeronautical convention. Due to the existence of two different coordinates frames one needs the transformation matrix between these two systems [24].

3.1.1 Rotation Matrix

The rotation vector for the positions x, y, and z are

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \tag{3.1}$$

$$R_y = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{3.2}$$

$$R_z = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

Then the consequent rotations around $z \to y \to x$ axis can be described by the rotation matrix also known as direction cosine matrix.

$$D = R_x R_y R_z \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\cos\theta \end{bmatrix} \tag{3.4}$$

The D matrix is orthogonal ($D^{-1} = D^T$) and describes the rotation matrix from the inertial frame to the body frame. In the body frame the linear velocity is defined by the nomenclature $\mathbf{V}_B = [u, v, w]^T$ $(m.s^{-1})$ and the linear velocity is $w = [p, q, r]^T$ $(rad.s^{-1})$. The relation between the linear velocity in the body frame and the position of the quadrotor in the inertial frame is defined by the following equation.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = D^{-1} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{3.5}$$

3.1.2 Transformation matrix for angular velocities

The transformation matrix for angular velocities in the inertial frame to body frame is given by equation [9]

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = E \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{3.6}$$

where

$$E = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix} \tag{3.7}$$

Then the second set of state equations rotating attitude with angular velocity in the body frame is

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = E^{-1} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{3.8}$$

3.1.3 Linear acceleration

The linear acceleration in the inertial frame is described by the Newton's Second Law

$$\mathbf{F} = m\dot{\mathbf{V}} \tag{3.9}$$

where m is the quadrotor's mass which is constant and V is the velocity vector in the body frame. The speeds u, v and w are measured in body frame coordinates and the body frame velocity vector can rotate and change its magnitude at the same time. This leads to total derivative of vector V ([5], p.10).

$$\mathbf{F} = m\dot{\mathbf{V}} + w \times m\mathbf{V} \tag{3.10}$$

Substituting the **V** and w vector by the corresponding coordinates

$$\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} + m \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{3.11}$$

After expanding the cross product and reorganizing

$$\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m \begin{bmatrix} \dot{u} + qw - rv \\ \dot{v} + ru - pw \\ \dot{w} + pv - qu \end{bmatrix} \tag{3.12}$$

The aerodynamic forces here are not taken into consideration. This leaves us with the trust of the propellers T and the weight force W as the only external forces acting in the quadrotor.

$$\begin{bmatrix} W_x \\ W_y \\ W_z - T \end{bmatrix} = m \begin{bmatrix} \dot{u} + qw - rv \\ \dot{v} + ru - pw \\ \dot{w} + pv - qu \end{bmatrix} \tag{3.13}$$

The weight force is always acting in the inertial frame in the z axis. Conversion to body frame is done by the rotation matrix (equation 3.4).

$$D \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} = m \begin{bmatrix} \dot{u} + qw - rv \\ \dot{v} + ru - pw \\ \dot{w} + pv - qu \end{bmatrix} \tag{3.14}$$

After reorganizing

$$\begin{aligned} \dot{u} &= rv - qw - g\sin\theta \\ \dot{v} &= pw - ru + g\cos\theta\sin\psi \\ \dot{w} &= qu - pv + g\cos\psi\cos\theta - \frac{T}{m} \end{aligned} \tag{3.15}$$

Considering no motor dynamics the thrust of all rotors is (”thrust is proportional to the square of the propeller’s angular rate” ([16], p.8))

$$T = b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \tag{3.16}$$

Where b is a trust coefficient and Ω_i is speed of each rotor. We can now substitute equation 3.15 and get the following expression.

$$\begin{aligned}\dot{u} &= rv - qw - g\sin\theta \\ \dot{v} &= pw - ru + g\cos\theta\sin\psi \\ \dot{w} &= qu - pv + g\cos\psi\cos\theta - \frac{b}{m}(\Omega_1^2 + \Omega_2^2 + \Omega_3^3 + \Omega_4^4)\end{aligned} \tag{3.17}$$

3.1.4 Angular acceleration

Application of an external torque **M** will change the angular momentum **H** of the quadrotor. But the angular momentum vector changes its direction hence the total derivative of vector H is applied ([5], p.13).

$$\mathbf{M} = \dot{\mathbf{H}} + w \times \mathbf{H} \tag{3.18}$$

And

$$\mathbf{H} = \mathbf{I}w, \tag{3.19}$$

where w is the change of the attitude and I is the moment of inertia of the quadrotor. The quadrotor is a rigid body symmetric about its xz and yz plane, and the rotation axes coincidences with the principal axes, then the moment of inertia tensor is

$$\mathbf{I} = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \tag{3.20}$$

Then

$$\mathbf{M} = \mathbf{I}\dot{w} + w \times \mathbf{I}w \tag{3.21}$$

After expanding

$$\begin{aligned} M_x &= \dot{p}I_x + qr(I_z - I_y) \\ M_y &= \dot{q}I_y + pr(I_x - I_z) \\ M_z &= \dot{r}I_z + pq(I_y - I_x) \end{aligned} \tag{3.22}$$

and because of the xz and yz symmetry

$$I_x \approx I_y \tag{3.23}$$

the equations can be simplified to

$$\begin{aligned} M_x &= \dot{p}I_x + qr(I_z - I_y) \\ M_y &= \dot{q}I_y + pr(I_x - I_z) \\ M_z &= \dot{r}I_z \end{aligned} \tag{3.24}$$

The external torques are produced by the trust and drag of the propellers. Neglecting the propeller's inertia and aerodynamic torques, then the external torques can be written as

$$\begin{aligned} M_x &= lb(\Omega_2^2 - \Omega_4^2) \\ M_y &= lb(\Omega_1^2 - \Omega_3^2) \\ M_z &= d(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 + \Omega_3^2) \end{aligned} \tag{3.25}$$

where d is the drag factor of the rotors and l is the distance of the propeller from the center of mass. So then the last set of equations of motion are

$$\begin{aligned} \dot{p} &= \frac{lb}{I_x}(\Omega_2^2 - \Omega_4^2) - qr\frac{I_z - I_y}{I_x} \\ \dot{q} &= \frac{lb}{I_y}(\Omega_1^2 - \Omega_3^2) - pr\frac{I_x - I_z}{I_y} \\ \dot{r} &= \frac{d}{I_z}(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 + \Omega_3^2) \end{aligned} \tag{3.26}$$

3.2 Robot Operating System and Gazebo Simulator

Robot Operating System (ROS) is a open source robotic middleware that provides driver, communication tool, library, and package for various type of robots [6], [17], [23], [25], [33]. Published in 2007, it is now used by more than 175 institutions worldwide. It supports both C++ and Python the former being the language chosen for this thesis.

ROS unique architecture it's one of the reason of its worldwide acceptance. Programmers can adapt code from different robots without concerning about the low level control of each individual robot. For example, in a project where the goal is to determine the robot's localization system, the programmer doesn't need to worry about the sensory readings because it has been handled by ROS. A feature that concerns us deeply since the purpose of this thesis is to develop a high-end controller and the low level properties are not to be taken into account in the process.

The policy to make ROS open source also enables collaborative work between people around the world. It is possible to use work developed by other people as well as share our own to anyone who wants to continue working on it or adapt it for other purposes. This allows us to focus on our work without the need to waste time developing software that was already invented or created in the past.

ROS also provides a robotic simulator that closely resembles the real world environment, namely Gazebo and R-Viz. In this environments we can create replicas of our robots, with all its properties, and simulate a world with all the characteristics we would need. We can use this simulation tool to prove the work we developed is feasible for a world with perfect conditions before applying it to the real robot in order to minimize the risk. A robotic system typically has any three-dimensional coordinate frames that change over time. The tf ROS package keeps track of multiple coordinate frames in the form of a tree structure. The tf package maintains the relationships between coordinate frames of points, vectors, and poses, and computes the transforms between them. The tf package operates in a distributed system; all ROS components within the system have access to information about the coordinate frames. The transform tree can also be viewed by developers for debugging by utilizing the $view_frames$ tool as shown in Appendix A.

3.2.1 Gazebo Simulator

Gazebo is one of the simulator features provided by ROS other than R-Viz. It provides many robot models which resemble the real one closely. In Gazebo, we can make our own model robot by writing out the mechanical properties as well as designing environment along with the furnitures and even the lighting condition, just like in Figure 3.2. Because of gaining popularity, Gazebo right now is a standalone project in (http://gazebosim.org). R-Viz is a 3D visualization environment that is part of the ROS visualization stack. It shows the tf's (shown in detailed in Appendix A) of all the robot parts, either fixed or mobile, as well as the tf of the inertial frame, add the respective tf of any static obstacles that may be present in the environment.

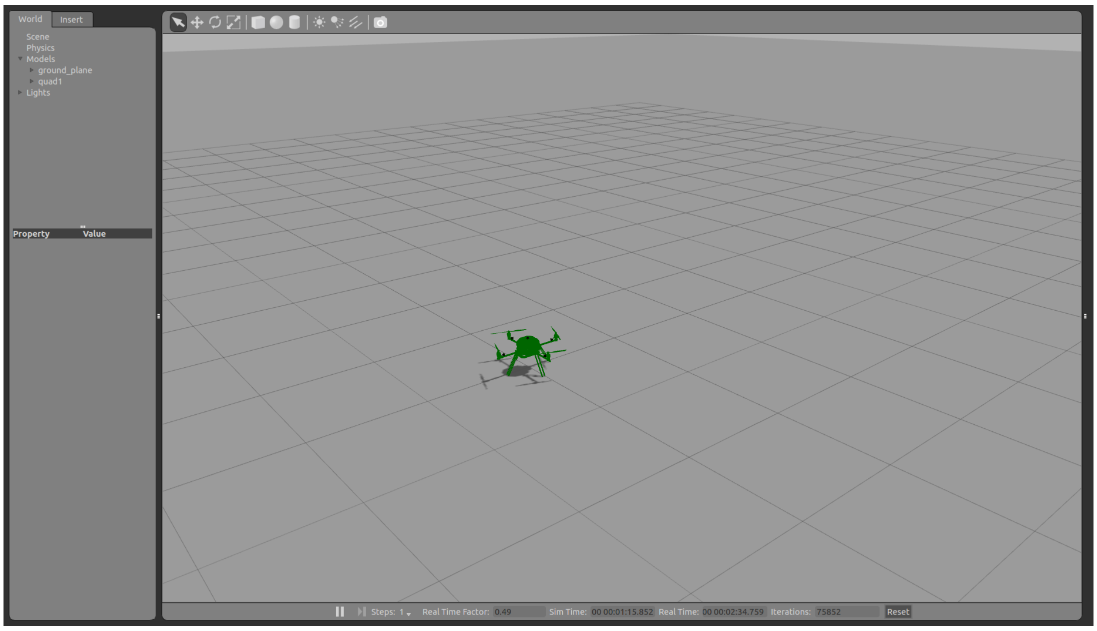


Figure 3.2: Gazebo Simulator with quadrotor model

The focus of this thesis is on path planning and formation problem, a higher level controller than the localization and mapping. By using Gazebo's global map coordinate, we can assume that all the information, including vehicle and obstacle locations, are known by every agent. However, it is also possible to create environments in which the position of the robot is not known in the world and only in relation to some obstacle. This is particularly interesting for some of the problems that are going to be tackled in this thesis.

3.2.2 Quadrotor model in Gazebo Simulator

The model used in Gazebo to represent the quadrotor has the specifications and dynamics of the robot available in the laboratory, as seen in Figure 3.2. The reason why we choose this quadrotor is because it is a model that was developed at the intelligent robots and systems group at ISR/IST this provides outputs that can be used by future students to work on their projects as well as a common ground for the research projects the group will take in the near future.

In order to develop a correct model of a quadrotor, libraries with the dynamics and properties of such a system is required. Hector [23] is a collection of ROS stacks originally developed by the Technische Universitat Darmstadt (Germany). These stacks supply several tools to simulate or to interact with robots. SLAM, location or modeling are some of them.

In this collection, the $hector\_quadrotor$ stack is going to be utilized with that tool, it is going to be able to model (with the physical parameters) a 3D quadrotor model in Gazebo and to have a 3D visualization of the trajectory performed by the quadrotor when executing the path generator by STL solver.

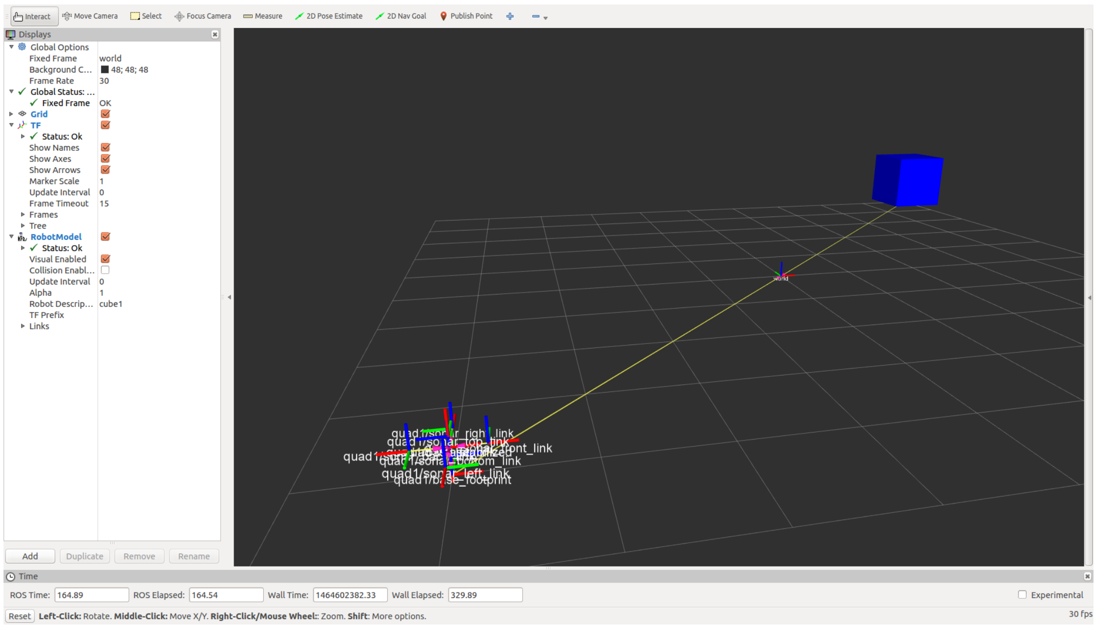
In Figure 3.3 we have an environment with one quadrotor created using the package discussed above and with the physical characteristics of the quadrotor from our laboratory. The corresponding frame tf's of the quadrotor, from the static object, and also the inertial frame tf are represented in Appendix A. Note that in R-Viz the inertial frame, that represents the origin of the environment and the absolute position of the quadrotor and the object, is called world frame by default.

This model is already defined with the all the topics necessary for the control of the position and attitude of the quadrotor. The topics we are focusing on are the cmd_vel and $ground\_to\_tf/pose$. Publishing to the topic cmd_vel we are able to send messages of linear velocity (m/s) and angular velocity (rad/s) to the respective linear and angular coordinates to increase or decrease the speed of the quadrotor. The $ground\_to\_tf/pose$ topic provides messages of the current position of the quadrotor, so subscribing to the topic informs us of the trajectory being taken.

3.3 Design of the Control System

The topic cmd_vel takes velocity commands as messages. The controller generated by the STL solver gives trajectory points that correspond to the path the quadrotor needs to take in order to reach the desired goal. This goal positions are cartesians coordinates in the x, y, and z axis. To send these positions to the quadrotor in a way that the cmd_vel topic can understand a controller needs to be implement. The approach developed was a velocity PI controller in which the distance between the current position and the goal position is the error to minimize and the output is the velocity [1]. A diagram of a velocity PID controller can be seen in Figure 3.4.

This type of controllers are commonly used for motor type controllers. This is why we choose this



```
Global Options
  Fixed Frame          world
  Background C...      48; 48; 48
  Frame Rate           30
✓ Global Status: ...
  ✓ Fixed Frame        OK
Grid                   ✓
TF                     ✓
  ✓ Status: Ok
  Show Names           ✓
  Show Axes            ✓
  Show Arrows          ✓
  Marker Scale         1
  Update Interval      0
  Frame Timeout        15
  Frames
  Tree
RobotModel             ✓
  ✓ Status: Ok
  Visual Enabled       ✓
  Collision Enabl...   ☐
  Update Interval      0
  Alpha                1
  Robot Descrip...     cube1
  TF Prefix
  Links

ROS Time: 164.89   ROS Elapsed: 164.54   Wall Time: 1464602382.33   Wall Elapsed: 329.89

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.
```

Figure 3.3: Quadrotor model shown in R-Viz with object. This is the Graphical User Interface (GUI) of R-Viz where it shows the quadrotor and object's tf's as well as a representation of the object structure. All the quadrotor tf's can be seen in more detail in Appendix A.

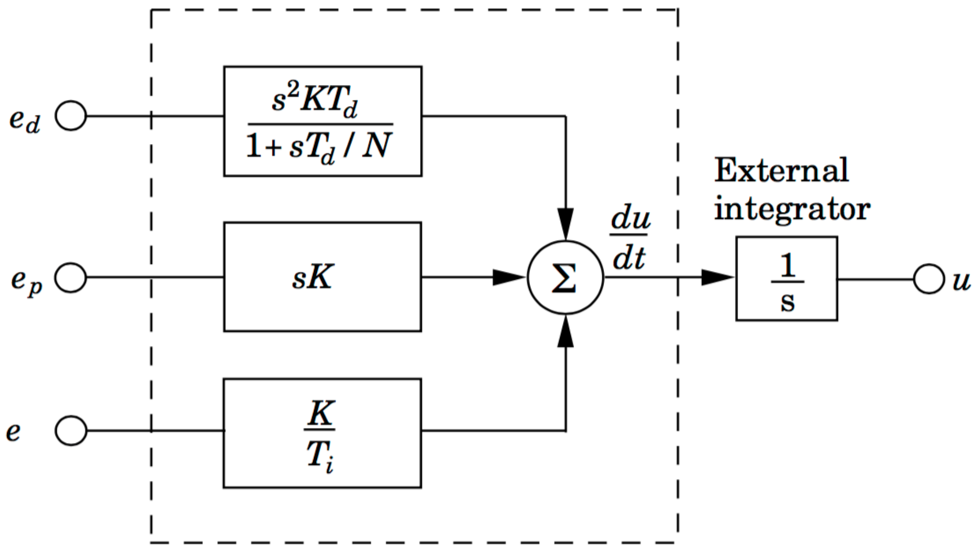


Figure 3.4: Block diagram of a PID algorithm in velocity form. [1]

controller for the quadrotor propellers. To note that our controller doesn't require a derivative part as our process reacts well with only the proportional and integral part as we can see in this step size example ran in the MATLAB simulation (Figure 3.5) as well as in the quadrotor path represented in Gazebo which will be shown further on in this thesis. This simulation in MATLAB is a simplified representation of the process in ROS where the $ground\_to\_tf/pose$ messages of the current position and the goal positions are the inputs provided, and the cmd_vel messages are the outputs calculated by the PI controller. There is no quadrotor non linear model implemented in the PI controller as the dynamics of the model are dealt inside the Gazebo simulator when the velocity commands from the cmd_vel topic are received.

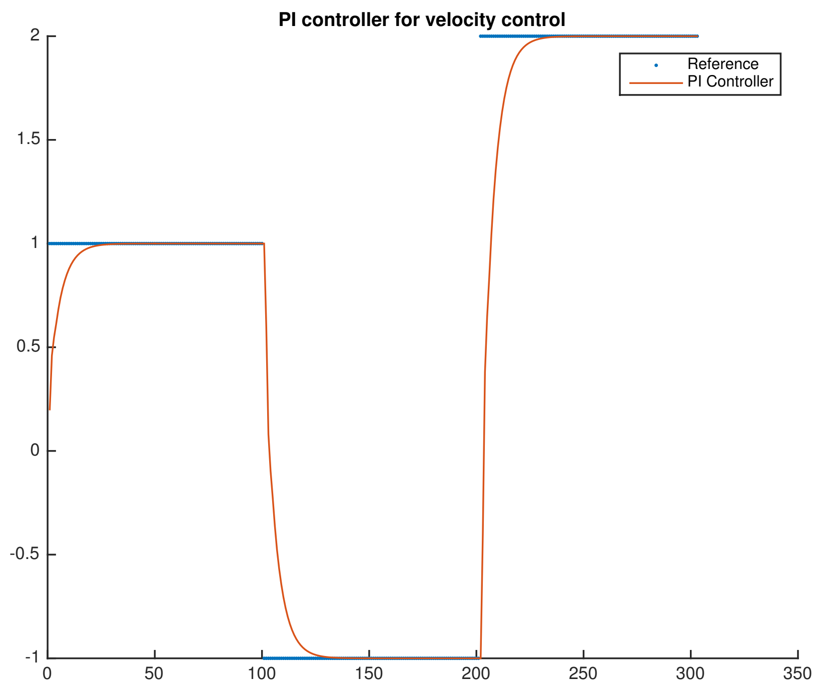


Figure 3.5: Step size output of the PI controller in velocity form

The PI parameters chosen are the following

$$K_p = 10 \quad K_i = 1 \quad T_i = 0.001$$

this is because a higher controller gain K_p and a lower integral time T_i give a more proportional and integral action which combined allow for a faster control. This PI controller is applied to all velocity coordinates individually since the goal is to minimize the distance between the current position of the cartesian coordinate and the goal position of the same cartesian coordinate, as all coordinates are independent of each other. A node called $Quad\_node$ is where the calculations and ROS messages are processed to give the quadrotor the right instructions to perform the desired trajectory. An illustration of the graph with all the nodes and topics used for the example in Figure 3.3 is shown in Figure 3.6.

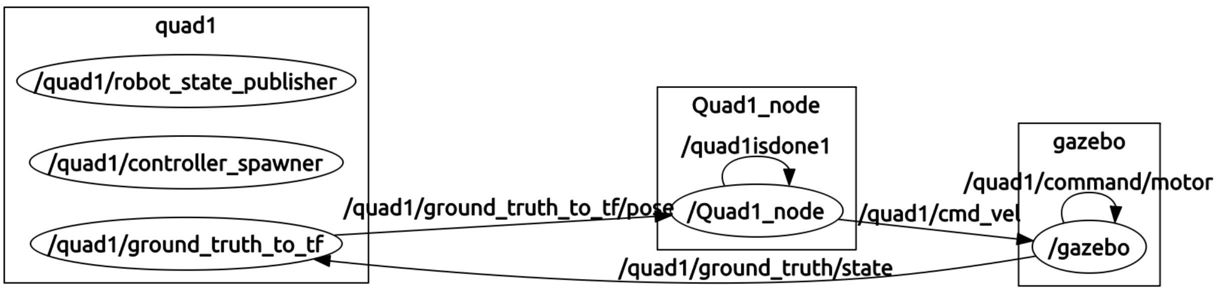

quad1
/quad1/robot_state_publisher
/quad1/controller_spawner
/quad1/ground_truth_to_tf
/quad1/ground_truth_to_tf/pose
Quad1_node
/quad1isdone1
/Quad1_node
/quad1/cmd_vel
gazebo
/quad1/command/motor
/gazebo
/quad1/ground_truth/state

Figure 3.6: Graph of 1 Quad

Chapter 4

Numerical Solution of STL Constraints

In this chapter a succinct explanation of the Mixed-Integer Linear Programming (MILP) encoding of STL specifications will be given. A small example will be solved in detail to give a better insight of the software used in this work. In Appendix B we include a full detail explanation of all the steps taken for the encoding of Mixed-Integer Linear (MIL) constraints of the STL specifications.

4.1 STL specification for integrator model

To give a better explanation of the process of generating mixed integer linear constraints, the focus will be in solving a problem with an integrator model. The purpose of using an integrator model ($\dot{x}_1 = u_1; \dot{x}_2 = u_2$) in contrast with the desired quadrotor model is for simplicity and easier comprehension of the problem. For exemplification of the process an integrator model works similarly to using a more complex model. In this problem, the aim is to keep the vehicle between a two dimensional environment of size 20×20 and at the same time avoid an obstacle of size 5×5 centered at the point $(x, y) = (7.5, 7.5)$.

$$\begin{aligned} \Box_{[0,\infty]}((x_1(t) < 5 \; or \; x_1(t) > 10 \; or \; x_2(t) < 5 \; or \; x_2(t) > 10) \\ and \;\; (x_1(t) > 0 \; and \; x_1(t) < 20 \; and \; x_2(t) > 0 \; and \; x_2(t) < 20)) \end{aligned} \tag{4.1}$$

Section B.2 from Appendix B explains the encoding of STL formulas into MIL constraints. It starts by defining the formula φ , as shown in Equation (4.2), consisting of all the specifications. The notation $\vee$ represents the logical 'or', and $\wedge$ is the logical 'and'.

$$\begin{aligned} \varphi = (x_1(t) < 5 \vee x_1(t) > 10 \vee x_2(t) < 5 \vee x_2(t) > 10) \\ \wedge \, x_1(t) > 0 \wedge x_1(t) < 20 \wedge x_2(t) > 0 \wedge x_2(t) < 20 \end{aligned} \tag{4.2}$$

Given a formula φ, a robustness variable ρ_t^{φ} (Section 2.4.3) is introduced whose value is tied to a set of MIL constraints. This leeds to a variable r_t^{φ} as explained in section B.3 from Appendix B.

4.2 Mixed Logical Dynamical Systems

Mixed Logical Dynamical Systems (MLD) is a class of hybrid systems in which logic, dynamics and constraints are integrated. This lead to a description in the form

$$\begin{aligned} x(k+1) &= A \cdot x(k) + B_1 \cdot u(k) + B_2 \cdot \delta(k) + B_3 \cdot z(k) \\ y(k) &= C \cdot x(k) + D_1 \cdot u(k) + D_2 \cdot \delta(k) + D_3 \cdot z(k) \\ E_1 \cdot x(k) &+ E_2 \cdot u(k) + E_3 \cdot \delta(k) + E_4 \cdot z(k) \leq g_5 \end{aligned} \tag{4.3}$$

where $x(k) = [x'_c(k)\ x'_l(k)]'$, $x_c(k) \in \mathbb{R}^{n_c}$ and $x_l(k) \in \{0,1\}^{n_l}$ (y(k) and u(k) have a similar decomposition), and where $z(k) \in \mathbb{R}^{r_c}$ and $\delta(k) \in \{0,1\}^{r_l}$ are auxiliary variables. A, B_i, C, D_i and E_i denote real constant matrices and g_5 is a real vector. The inequalities have to be interpreted component-wise. In the case of the integrator model and also the quadrotor model the matrix B_2, B_3, D_2 and D_3 are zero because there is no need of auxiliary variables to define the dynamics of the model.

Going back to equation (4.2), every predicate can be represented by its own auxiliary variable r_t, just as r_t^{φ} represents the auxiliary variable of φ.

$$\begin{aligned} r_t^{x_1(t)<5} &= x_1(t) - 5; & r_t^{x_1(t)>0} &= x_1(t); \\ r_t^{x_1(t)>10} &= 10 - x_1(t); & r_t^{x_1(t)<20} &= 20 - x_1(t); \\ r_t^{x_2(t)<5} &= x_2(t) - 5; & r_t^{x_2(t)>0} &= x_2(t); \\ r_t^{x_2(t)>10} &= 10 - x_2(t); & r_t^{x_2(t)<20} &= 20 - x_2(t); \end{aligned} \tag{4.4}$$

It is important to first introduce the concept of robustness here. Quantitative or robust semantics define a real-valued function r_t^{φ} of signal **x** and t implies that $(\mathbf{x}, t)$ satisfies φ. Looking at Equation (4.2) there is the disjunction ('$\vee$') of constraints, which can be described as the maximum of all the terms, and the conjunction ('$\wedge$') of constraints which can be interpreted as the minimum of all the terms. Using the auxiliary variable in (4.4) and the robustness satisfaction defined in section 2.4.3, the r_t^{φ} which represents the complete STL formula, can be defined as follows

$$\begin{aligned} r_t^{\varphi} = \min(\max(r_t^{x_1(t)<5}, r_t^{x_1(t)>10}, r_t^{x_2(t)<5}, r_t^{x_2(t)>10}), \\ r_t^{x_1(t)>0}, r_t^{x_1(t)<20}, r_t^{x_2(t)>0}, r_t^{x_2(t)<20}) \end{aligned} \tag{4.5}$$

This can be simplified by defining an extra variable from the outcome of the maximum operator, $r_t^1 = \max(r_t^{x_1(t)<5}, r_t^{x_1(t)>10}, r_t^{x_2(t)<5}, r_t^{x_2(t)>10})$. Since for each predicate μ there is an associated auxiliary boolean variable δ, which equals 1 if μ holds at time t, and 0 otherwise. For every auxiliary r_t variable defined there is a boolean variable δ represented by the variable P_t. In the case of variables $P_t^{x_1(t)>0}, P_t^{x_1(t)<20}, P_t^{x_2(t)>0}$ and $P_t^{x_2(t)<20}$ corresponding to robustness satisfaction variables $r_t^{x_1(t)>0}$, $r_t^{x_1(t)<20}, r_t^{x_2(t)>0}$, and $r_t^{x_2(t)<20}$, respectively, they equal 1 if and only if each corresponding predicate holds at time t. To exemplify, for predicate $x_1(t) > 0$ it has robustness satisfaction (and auxiliary variable) $r_t^{x_1(t)>0}$, which has boolean auxiliary variable $P_t^{x_1(t)>0}$ that is equal to 1 at time t if and only if x_1 at time

t is in fact larger than 0. The same applies to all other variables.

$$\begin{aligned}
P_t^1 = P_t^{x_1(t)<5} + P_t^{x_1(t)>10} = 1\\
P_t^2 = P_t^{x_2(t)<5} + P_t^{x_2(t)>10} = 1\\
P_t^1 = P_t^2 = P_t^{x_1(t)>0} = P_t^{x_1(t)<20} = P_t^{x_2(t)>0} = P_t^{x_2(t)<20} = 1
\end{aligned} \tag{4.6}$$

If it were the case of avoiding an obstacle, for example $\square_{[0,\infty]}\neg(x_1(t) > 5 \wedge x_1(t) < 10 \wedge x_2(t) > 5 \wedge x_2(t) < 10)$ then the boolean variables would be

$$\begin{aligned}
P_t^1 = P_t^{x_1(t)>5} + P_t^{x_1(t)<10} = 0\\
P_t^2 = P_t^{x_2(t)>5} + P_t^{x_2(t)<10} = 0\\
P_t^1 + P_t^2 = 0
\end{aligned} \tag{4.7}$$

The sum of the boolean variables have to be zero in order to respect the specifications. The boolean variables have to written in other to respect what is desired of the STL specifications.

4.3 Upper and lower bounds

Logical operation $\min$ is the conjunction of all the variables inside the minimum operator. As for $\max$ it is the disjunction of all the variables [37].

$$\begin{aligned}
r_t^1 &\geq r_t^{x_1(t)<5} & r_t^{\varphi} &\leq r_t^1\\
r_t^1 &\geq r_t^{x_1(t)>10} & r_t^{\varphi} &\leq r_t^{x_2(t)<20}\\
r_t^1 &\geq r_t^{x_2(t)<5} & r_t^{\varphi} &\leq r_t^{x_2(t)>0}\\
r_t^1 &\geq r_t^{x_2(t)>10} & r_t^{\varphi} &\leq r_t^{x_1(t)<20}\\
& & r_t^{\varphi} &\leq r_t^{x_1(t)>0}
\end{aligned} \tag{4.8}$$

To enforce that the constraints above are fulfilled, an upper and lower bound is added to the constraints. Following section B.3 from Appendix B it explains the robustness-based encoding adding the auxiliary boolean variables. Each Boolean operations are defined as:

Negation: $r_t^{\varphi} = \neg\, r_t^{\phi}$

Conjunction: $r_t^{\varphi} = \wedge_{i=1}^{m} r_{t_i}^{\psi_i}$

$$\begin{aligned}
\sum_{i=1}^{m} P_{t_i}^{\varphi_i} = 1\\
r_t^{\psi} \leq r_{t_i}^{\varphi_i}, i = 1\ldots, m\\
r_{t_i}^{\varphi_i} - (1 - P_{t_i}^{\varphi_i}) \cdot M \leq r_t^1 \leq r_{t_i}^{\varphi_i} + (1 - P_{t_i}^{\varphi_i}) \cdot M
\end{aligned} \tag{4.9}$$

where binary variables $P_{t_i}^{\varphi_i}$ are introduced for $i = 1, \ldots, m$, and M is a sufficiently large positive number.

For a more detailed explanation see [28]. These bounds ensures that there is one and only one $j \in \{1, \dots, m\}$ such that $P_{t_i}^{\varphi_i} = 1$, which ensures that r_t^{ψ} is smaller than all $r_{t_i}^{\varphi_i}$, and enforces that $r_t^{\psi} = r_{t_i}^{\varphi_i}$ if and only if $P_{t_i}^{\varphi_i} = 1$. Together, these constraints enforce that $r_t^{\psi} = \min_i(r_{t_i}^{\varphi_i})$. The encoding is similar for disjunctions but for $r_t^{\psi} = \max_i(r_{t_i}^{\varphi_i})$. It is now possible to define all the corresponding constraints.

$$\begin{aligned}
r_t^{x_1(t)<5} - (1 - P_t^1) \cdot M \leq r_t^1 \leq r_t^{x_1(t)<5} + (1 - P_t^1) \cdot M \\
r_t^{x_1(t)>10} - (1 - P_t^1) \cdot M \leq r_t^1 \leq r_t^{x_1(t)>10} + (1 - P_t^1) \cdot M \\
r_t^{x_2(t)<5} - (1 - P_t^1) \cdot M \leq r_t^1 \leq r_t^{x_2(t)<5} + (1 - P_t^1) \cdot M \\
r_t^{x_2(t)>10} - (1 - P_t^1) \cdot M \leq r_t^1 \leq r_t^{x_1(t)>10} + (1 - P_t^1) \cdot M \\
r_t^1 - (1 - P_t^1) \cdot M \leq r_t^{\varphi} \leq r_t^1 + (1 - P_t^1) \cdot M \\
r_t^{x_1(t)>0} - (1 - P_t^{x_1(t)>0}) \cdot M \leq r_t^{\varphi} \leq r_t^{x_1(t)>0} + (1 - P_t^{x_1(t)>0}) \cdot M \\
r_t^{x_2(t)>0} - (1 - P_t^{x_2(t)>0}) \cdot M \leq r_t^{\varphi} \leq r_t^{x_2(t)>0} + (1 - P_t^{x_2(t)>0}) \cdot M \\
r_t^{x_1(t)<20} - (1 - P_t^{x_1(t)<20}) \cdot M \leq r_t^{\varphi} \leq r_t^{x_1(t)<20} + (1 - P_t^{x_1(t)<20}) \cdot M \\
r_t^{x_2(t)<20} - (1 - P_t^{x_2(t)<20}) \cdot M \leq r_t^{\varphi} \leq r_t^{x_2(t)<20} + (1 - P_t^{x_2(t)<20}) \cdot M
\end{aligned} \tag{4.10}$$

where M is a sufficiently large positive number. Now that we have all the constraints of our example it is possible to re-write our model in a structured way using constraint matrices. Notice that our model is composed of equality constraints and inequality constraints. Here, $x(k)$ is the discrete states of the system, $u(k)$ is the control input, δ(k) is the vector of the binary variables, and $z(k)$ is the vector of the predicate variables. Using the MLD system from section (4.2) the inequality and equality constraints can be completed.

$$\begin{aligned}
E_1 \cdot x(k) + E_2 \cdot u(k) + E_3 \cdot \delta(k) + E_4 \cdot z(k) \leq g_5 \\
E_{1_{eq}} \cdot x(k) + E_{2_{eq}} \cdot u(k) + E_{3_{eq}} \cdot \delta(k) + E_{4_{eq}} \cdot z(k) = g_{5_{eq}}
\end{aligned} \tag{4.11}$$

The state-space model of a continuous time system without any disturbances is represented as follows

$$x(t) = A \cdot x(t-1) + B_u \cdot u(t-1) \tag{4.12}$$

The equalities in Equation (4.4) will be then replaced by the correct terms using (4.12).

$$\begin{aligned}
r_t^{x_1(t)<5} = A \cdot x_1(t-1) + B_u \cdot u(t-1) - 5 \\
\dots
\end{aligned} \tag{4.13}$$

$x_1(t)$ is the row elements of the state x_1 and $u(t-1)$ is the first row elements multiplied by the control inputs. Notice that $x(t-1)$ is the current states of the model and $u(t-1)$ the current input of the system. The next step is to convert MLD system to MLD-MPC problem with N_p as the prediction horizon. The only thing remaining is to define the bounds on the variables and create the cost function to give to the solver. **w** is the vector of all decisions variables in the MPC optimization problem composed of vectors $\tilde{u}$, $\tilde{\delta}$ and $\tilde{z}$. Since the output desired to calculate the cost function is the robustness function of the whole STL formula then the matrix C which is the output matrix depending on the state $x(t)$ will have all the parameters 0 except the one corresponding to r_t^{φ}.

$$\mathbf{w} = \begin{bmatrix} \tilde{u} \\ \tilde{\delta} \\ \tilde{z} \end{bmatrix}_{(n_u+n_\delta+n_z)\cdot N_p \times 1} = \begin{bmatrix} \tilde{u_1} \\ \tilde{u_2} \\ P_t^{x_1<5} \\ \dots \\ P_t^1 \\ r_t^{\varphi} \\ r_t^{x_1 \le 5} \\ \dots \\ r_t^1 \end{bmatrix}_{N_p \times 1} => C = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \dots \\ 0 \\ 1 \\ 0 \\ \dots \\ 0 \end{bmatrix}_{N_p \times 1} \tag{4.14}$$

The cost function is equal to the robustness function for the entire prediction horizon. Since the objective is to maximize the robustness function, a minus sign is added to the beginning of the equation.

$$J(t) = -\sum_{j=0}^{Np-1} r_{t+j}^{\varphi} \tag{4.15}$$

Hence the optimization problem can be defined as follows.

$$\begin{aligned} & \min(\mathbf{w}J(t)) \\ s.t. \quad & \text{equality constraints} \\ & \text{inequality constraints} \end{aligned} \tag{4.16}$$

The control methodology that is applied here is called model predictive control (MPC). In MPC, at each iteration, the optimal control sequence is computed over a finite horizon N_p. MPC uses the receding horizon principle, which means that after computing the optimal control sequence, only the first element is implemented in the current iteration. Subsequently, the horizon is shifted by one time step, and the optimization restarted to include any new information, such as about the disturbance measurements. To solve the above optimization problem, a Gurobi solver in MATLAB was used, which requires you to give the objective function and a function of constraints. Gurobi is optimization solver available for students that is compatible for MATLAB. If the result is feasible and bounded then the output will be the vector with the respective values. To run the simulation this is required to be solved as many times as the duration of the simulation, to do so the current state position and current input control given by the solver should be implemented in the model to calculate the next states and to introduce the initial state as now the new current state.

This problem solves a really simple environment without any time-dependent constraints. For a mission involving UAV's visiting goals, in this case objects to collect, it requires more complex specifications with the dynamics of the UAV but the principle is the same.

4.4 STL solver properties

In this section we provide the mandatory properties for the solver to find a feasible trajectory for a specific mission.

Choosing a model with the quadrotor dynamics is unnecessary because what we want from the controller are the trajectory positions generated by the solver to respect the STL specifications. So, giving the quadrotor model as information to the system is not only unnecessary but also computationally more demanding to the solver, due to the more complex dynamics. Instead, we choose to use a very simple model to represent the quadrotor. Since the dynamics are defined in the Gazebo simulator and the goal is to obtain the Cartesian coordinates we choose a 3D integrator model to define the quadrotor in the STL solver.

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B_u = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 10 \end{bmatrix} \tag{4.17}$$

This way every input only controls one direction, and we privilege the z axis since we want the quadrotor to be as high from the ground as possible with little effort. For the case of a multi-vehicle mission with 3 quadrotors we just increase the number of variables from 3 to 9, 3 dimensions for each vehicle.

$$A = \begin{bmatrix} 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \end{bmatrix} \quad B_u = \begin{bmatrix} 1&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0 \\ 0&0&10&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&10&0&0&0 \\ 0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&10 \end{bmatrix} \tag{4.18}$$

With the respective state and input vectors:

$$x = [x_{1x}, x_{1y}, x_{1z}, x_{2x}, x_{2y}, x_{2z}, x_{3x}, x_{3y}, x_{3z}]$$

$$u = [u_{1x}, u_{1y}, u_{1z}, u_{2x}, u_{2y}, u_{2z}, u_{3x}, u_{3y}, u_{3z}]$$

where x_{ij} is the state space coordinate, i is the number of the quadrotor, j is the Cartesian coordinate, and u_{ij} is the input variable.

The A matrixes are the dynamics on all the inputs states x, Bu is the matrix dependent on the control inputs u. A matrix has dimension 9×9 and Bu 9×3. Additionally, the control inputs are represented over the course of the simulation time in discrete impulse with bounds of 10 $(m.s^{-1})$

$$-10 < u_{i,j} < 10 \tag{4.19}$$

for i, j = $\{1, 2, 3\}$. The cost function is a weight of 100 on the robustness of the specifications and a receding horizon of 10 steps.

$$J(t) = -\sum_{j=0}^{9} 100\, r^{\varphi}_{t+j} \tag{4.20}$$

Chapter 5

Case Studies

In this chapter we will show the results obtained when solving the multi-UAV mission coordination using STL specifications for the **Challenge 3** proposed by the International competition MBZIRC 2017. This challenge was divided into smaller problems each more complex than the previous one. The first case study will start by solving a mission with just one quadrotor and, given a specific goal, the vehicle would have to reach the destination and remain there until further orders. The next case study adds a new goal to the previous quadrotor. A static object will be introduced to the environment, the quadrotor would grab it and transport it to a second location to drop it. After this has been achieved, the complexity of the mission will be increased by introducing two more quadrotors. Adding three quadrotors to coexist in the same environment is possible but not without some rules to prevent clashes and/or conflict. These rules will be discussed in this chapter as well as a possible solution for each of them.

5.1 Case Study 1: 1 quadrotor reaching one goal

In this first example the mission is quite simple, the objective is to reach a specific location from a standing point, the initial position of quadrotor. The description of the mission for an arbitrary location $(x, y, z) = (10.5, 5.5, 0)$ is described as follows

$$\Diamond_{[0,7]}\Box_{[0,\infty]}(x_1(t) > 10.5 \wedge x_1(t) < 11 \wedge x_2(t) > 5.5 \wedge x_2(t) < 6 \wedge x_3(t) > 0 \wedge x_3(t) < 0.5)$$

$$\Diamond_{[0,3]}(x_3(t) > 3 \wedge x_3(t) < 5)$$

with every coordinate having a threshold of 50 centimeters as an acceptable estimation error. The first STL specification is forcing the quadrotor model to reach during the first 8 time steps ($\Diamond_{[0,7]}$) the final destination as defined previously and remain there for the end of the simulation time hence the boolean operators $\Box_{[0,\infty)}$ to represent the whole specter of time, combining with the $\Diamond$ boolean operator, giving it the ”eventually always” effect. Since the model used is a simple 3 dimensional integrator without the second STL specification the trajectory generated would be a path relatively close to the ground since both the initial position and goal location are at the z axis level. Forcing the quadrotor to reach,

”eventually” a high altitude, would force the generated trajectory to create a parabolic path to the goal as it can be seen in Figure 5.1. However, this causes a weird behavior after reaching the goal location. The ”eventually” specification is to be valid between time $t \in [0, 3]$ time step. This is fulfilled at t_2 shown in Figure 5.1.

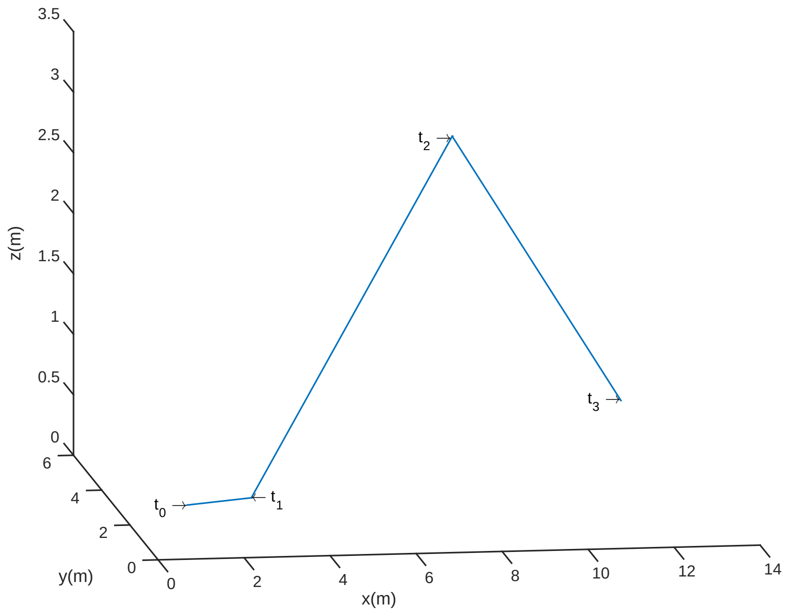


Figure 5.1: The quadrotor position trajectory for a mission with a single quadrotor reaching the location $(x, y, z) = (10.5, 5.5, 0)$

5.1.1 ROS Implementation

After generating the path from the STL solver the state space coordinates resulting from the control inputs are output and imported to the STLnode#, where # is the indication of the quadrotor. This node is where the goal positions are read and sent to the Quad#_node to be computed to velocity inputs using the PI controller. STLnode# needs to synchronize with Quad#_node to only send the current goal position when it has reached the previous goal. There is where the topic quad#chatter comes into use to notify each other about the node status (see Figure 5.2 to see the connections between nodes).

All this combined is visualized in Gazebo and R-Viz simulator where the quadrotor tfs are represented. A frame by frame representation of the generated path from the STL solver, now performed by our quadrotor model, can be visualized in Figure 5.3. The PI controller follows correctly the path and the PI parameters used for the simulation make a smooth trajectory with a small enough error to the goal positions. In the next section we will continue from this point by adding an object to the environment.

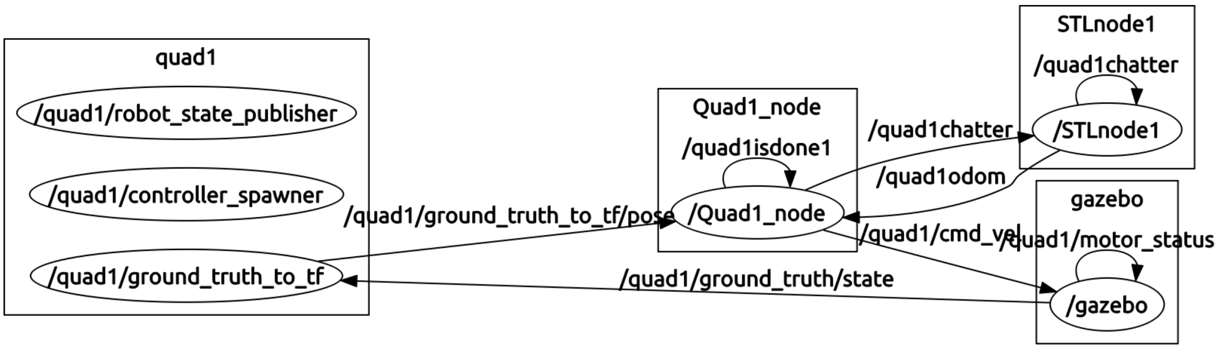


Figure 5.2: All topics and nodes for 1 quad mission without object

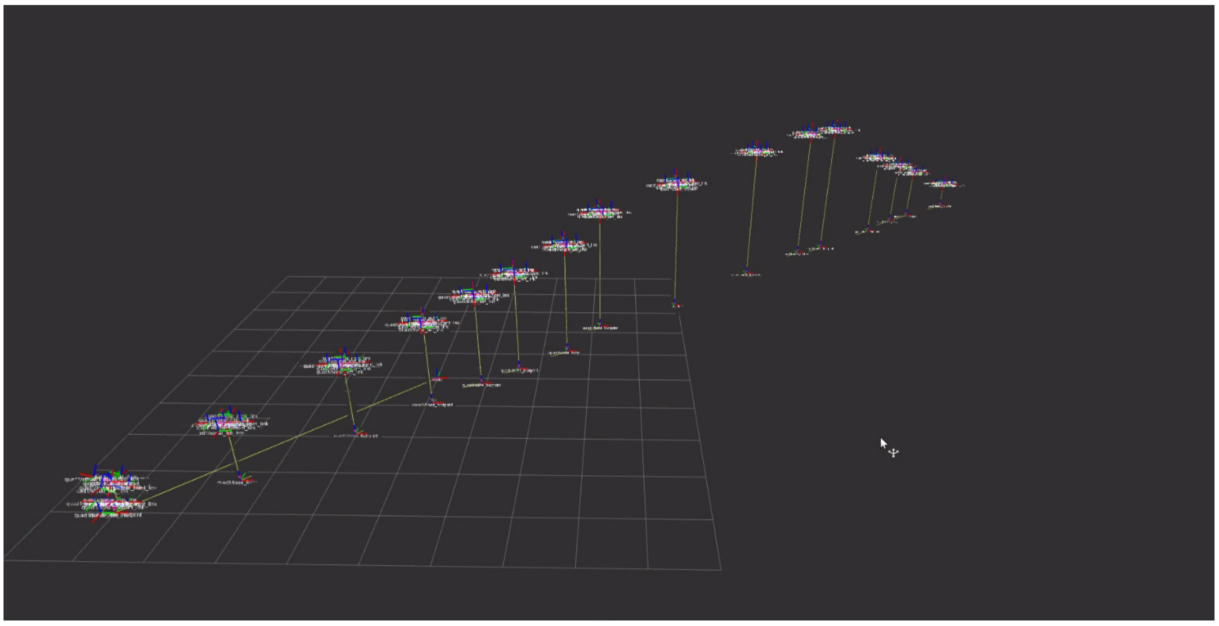


Figure 5.3: 1 Quad reaching goal position 1

5.2 Case Study 2: 1 quadrotor grabbing an object

For the next Case Study, an object is added to the environment. The goal is for the quadrotor to reach the object location, directly above it, and grab the object for future instructions. For simplicity we just create a box with dimension $0.5 \times 0.5 \times 0.5$.

5.2.1 ROS Implementation

This object will be static and the center of mass position is defined by the user when running the node tf_cube#. This node is responsible for the relative position of the static object in the world. It receives information from both Quad#_node and STLnode# whether its respective goals have been reached. This is to confirm that the quadrotor has received the goal position located above the object in order to proceed with the grasping. A full description of the nodes and topics can be seen in Figure 5.4. If Quad#_node has reached the final goal position of the first path, this means that the quadrotor has performed the trajectory and has reached the static object. As so, a notification is send to tf_cube# and the relative position of the object has changed from a specific location in the world to be at a certain distance relative to the quadrotor, just below the quadrotors's tf $base\_link$ (for more information into the quadrotor tf's check Appendix A.1). As a result the object, since its tf is linked to the quadrotor's tf, it follows the trajectory of the quadrotor giving it the sense of being transported. On the other hand,

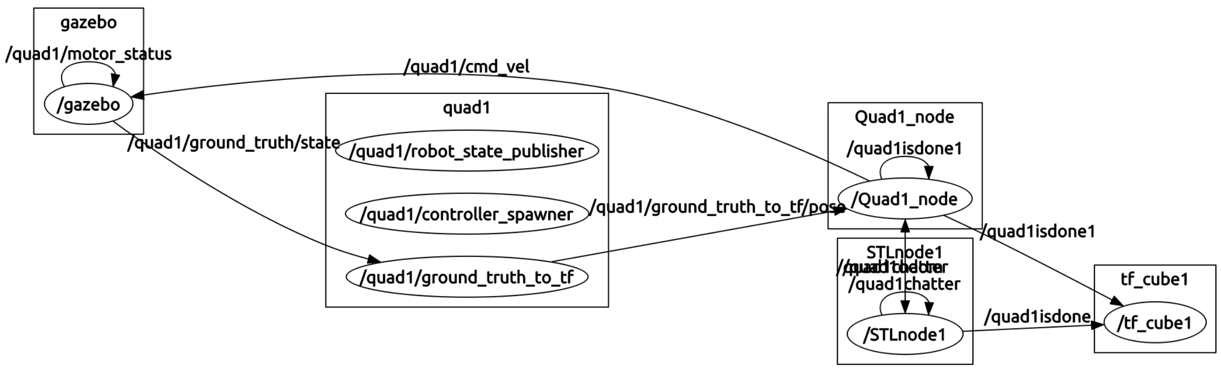


Figure 5.4: All topics and nodes for 1 quad mission with 1 object

if STLnode$\#$ send its information it means that the last goal position was sent and received, so the quadrotor is situated at the location to drop the object. In this case, the $tf\_cube\#$ is positioned in the desired location where the object needs to be dropped relative to the world.

A visualization of the trajectory generated by the STL solver is represented in Figure 5.5 with the static object being grabbed and moved to the goal location. The path is obtained with the same specifications as the ones referred in the section 5.1. The only needed modification is the location to drop the object which in this case was $(x, y, z) = (5.5, 3.5, 3)$, everything else is pre-defined to work for every single quadrotor mission to start from some initial state (defined in respect to every mission), and end at a destination goal. To note that this goal position is for the quadrotor model, as for the object, its final location is defined in the tf_cube$\#$ node, which is usually located below the quadrotor at ground level.

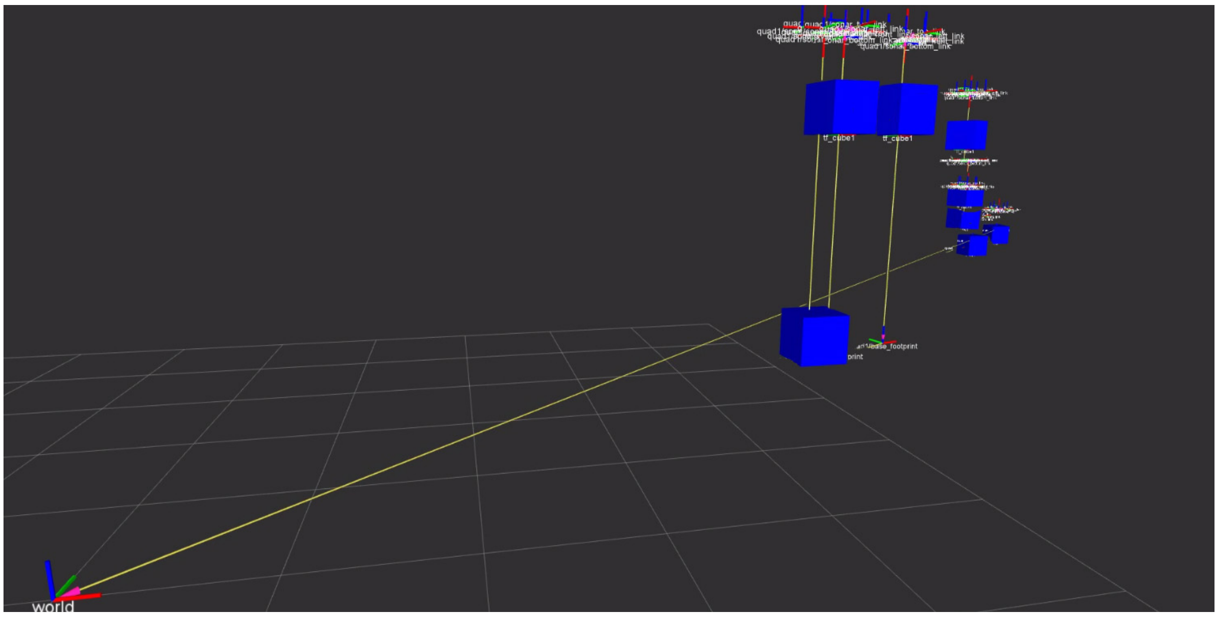


Figure 5.5: 1 Quad grabbing and moving object to goal position 2

This concludes the challenges involving only 1 quadrotor in a known environment. From this point onwards, the missions will be more complex as it will have 3 independent quadrotors navigating in the same environment. The objective is to try to replicate the same goals with just a single model but this time for a multi-vehicle mission where every model will have to find, grab and move an object to a specific location. As it will become clear in the next sections this will cause some different problems when trying to solve these objectives. Problems that are more interesting to solve due to its complexity and formulation. In return this gives us tremendous insight on the capabilities of using STL specifications when solving multi-vehicle mission collaboration.

5.3 Case Study 3: 3 quadrotors mission

At this stage, after solving problems regarding single quadrotor missions, the next objective would be to tackle more complex missions involving more than one quadrotor. This is why this thesis is based on the **Challenge 3** from the international competition MBZIRC 2017. It involves 3 quadrotors where autonomously each one will find, grab and move one object in an environment. To achieve it we will begin by replicating what we did for the single system for a multi-system. Although every model has the same properties and dynamics, each one will have a different class corresponding to each quadrotor. So, when choosing quadrotor 1 for example, all the tfs will be under the class name "$quad1/$". Below there is possible configuration of the environment with 3 quadrotors.

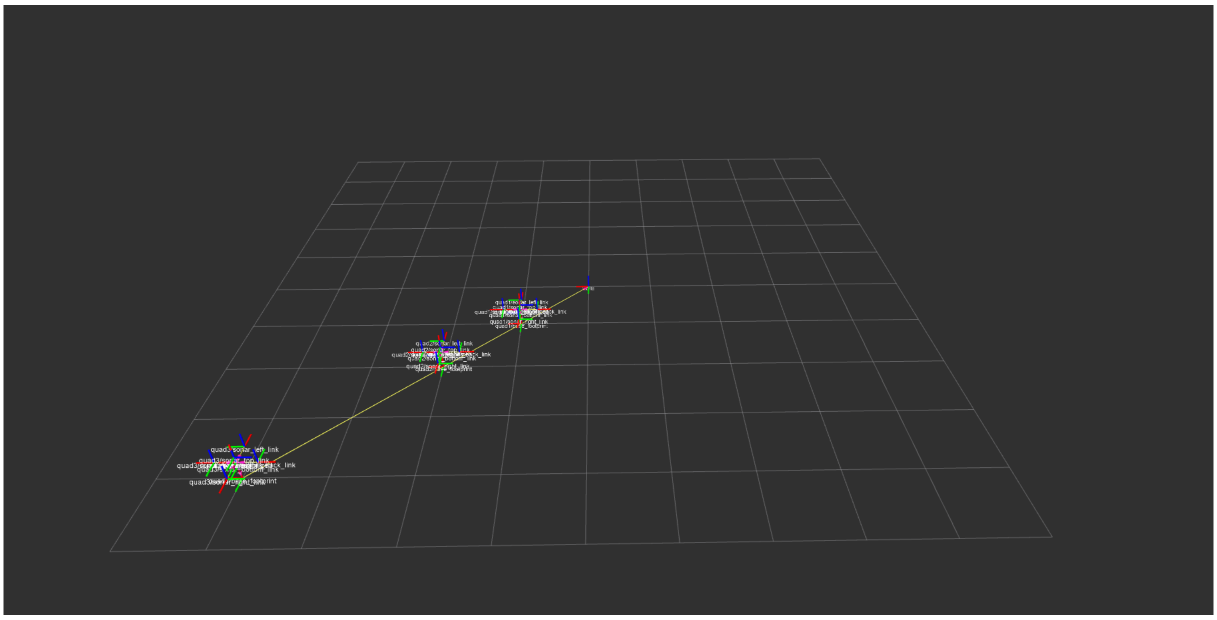


Figure 5.6: Gazebo simulator with 3 quadrotors

When defining the STL specifications in the STL solver the model will have to adapt to the complexity of the mission. In this case, since we have 3 quadrotors the model will be identical to the matrixes present in equation 4.18. This mission involves each quadrotor to grab a static object in the environment, and it is predisposed in a certain way that the path generated is not going to cross the other quadrotor's path.

$$\Box_{[0,\infty]}\Diamond_{0,10]}(x_1(t) > 10.5 \wedge x_1(t) < 11 \wedge x_2(t) > 15.5 \wedge x_2(t) < 16 \wedge x_3(t) > 0 \wedge x_3(t) < 0.5)$$
$$\Box_{[0,\infty]}\Diamond_{0,10]}(x_4(t) > 12.5 \wedge x_4(t) < 13 \wedge x_5(t) > 5.5 \wedge x_5(t) < 6 \wedge x_6(t) > 0 \wedge x_6(t) < 0.5)$$
$$\Box_{[0,\infty]}\Diamond_{0,10]}(x_7(t) > 17.5 \wedge x_7(t) < 18 \wedge x_8(t) > 15.5 \wedge x_8(t) < 16 \wedge x_9(t) > 0 \wedge x_9(t) < 0.5)$$

$$\Diamond_{[0,3]}(x_3(t) > 3 \wedge x_3(t) < 5)$$
$$\Diamond_{[0,3]}(x_6(t) > 3 \wedge x_6(t) < 5)$$
$$\Diamond_{[0,3]}(x_9(t) > 3 \wedge x_9(t) < 5)$$

5.3.1 ROS Implementation

The same applies to the ROS configuration. For every quadrotor there will be a Quad#_node, a STLnode#, and a tf_cube#. When all of this is defined we are ready to build a mission like the one seen in Figure 5.7. Every quadrotor has a specific static object to go different for each other. As we will discover in the next sections this a very particular problem that will unlikely happen very often. To give a more generic approach to the challenge, the mission needs to be less defined to just one situation.

This example results in a smooth, free collision mission, where every quadrotor performs its objective successfully. However, this isn't always the case so when a collision is eminent there must be a way to correct and avoid the problem. This is what we are going to the discuss in the next section as well as some other issues that will be mention further into this thesis.

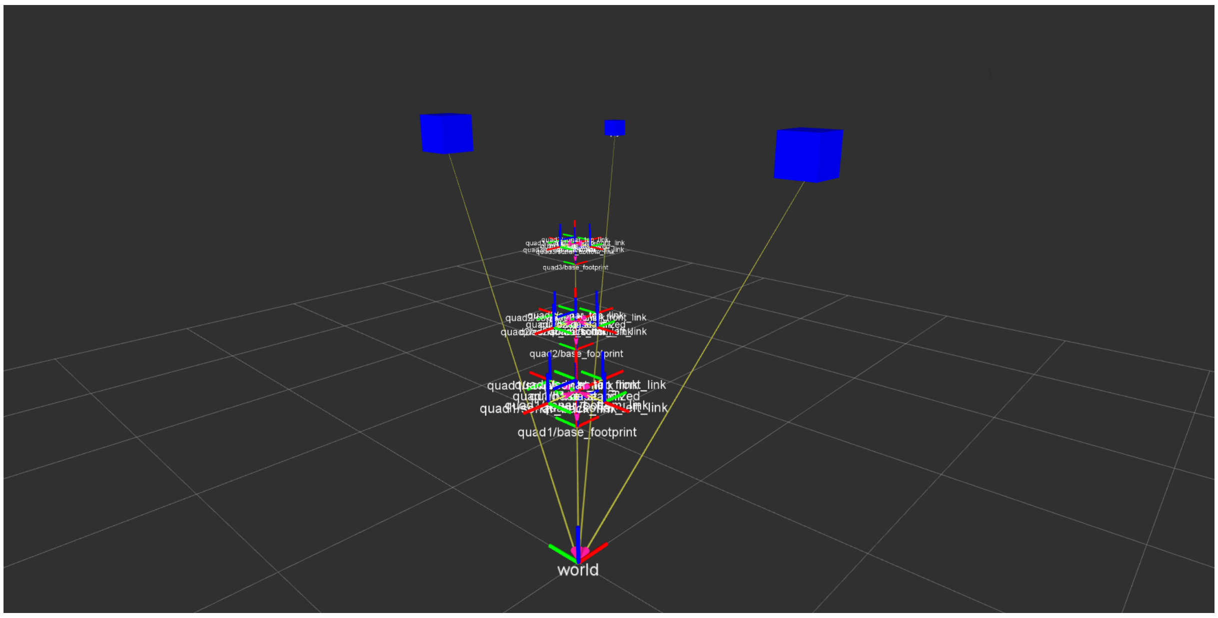


Figure 5.7: A mission with 3 quadrotors and with 3 static objects in the environment

The next sections we will discuss every problem individually, referring examples where this situation can cause problems to the possible trajectory generated. Each problem is properly introduced and explained, the following sections will propose some solutions to tackle them. Some problems will have more than one solution which have different outcomes and advantageous that are interesting to compare. In the end a mission which will simulate a possible solution for the **Challenge 3** will be represented and visualized in detail.

5.3.2 Problem 1: Collision

The first problem discusses when the trajectory generated for one or more quadrotors will, in a moment in time during the path, intersect with the path of another quadrotor. This, as a result, may cause the crash of two or more quadrotors and the consequences of this crash are dangerous and may result in arbitrary, spontaneous movements from the quadrotors that can end up in an unsuccessful mission. In Figure 5.8 we see an example of a mission where all three quadrotors have intersecting trajectories.

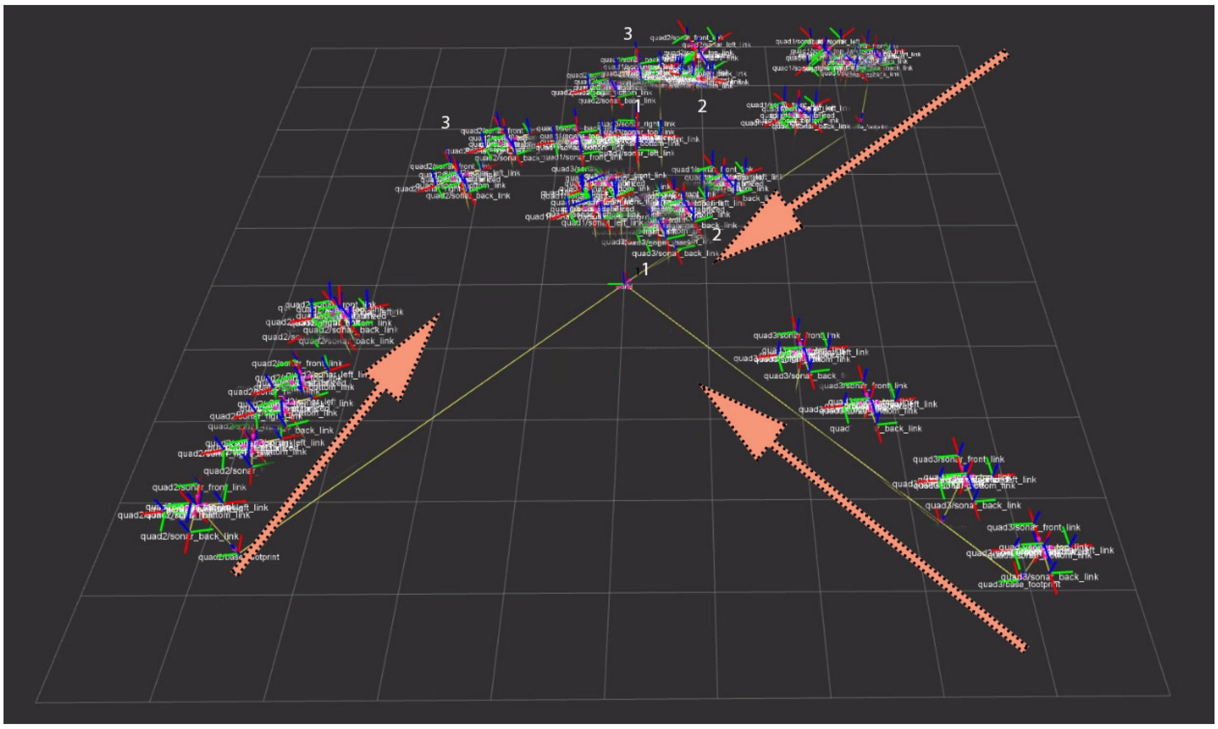


Figure 5.8: Collision problem when 3 quads have crossing paths. The first crash happens in the middle of the picture between quadrotor 1 and 2, and then a second crash occurs between quadrotors 1, 2 and 3, on the top of the picture. YouTube video: `https://www.youtube.com/watch?v=-I32qvEif0Q`

This specific example results in very pronounced collisions, the first one where the middle quadrotor tries to reach the center of the map and the right quadrotor is performing its trajectory crossing the same center of the map. Resulting in an uncontrolled trajectory of the two quadrotors rotating jointly due to the two opposite linear velocities of different absolute values. This rotating velocity leads to the second crash as the two quadrotors that collided moved to the path of the third quadrotor. The collision of the three quadrotors causes the excess computations of all the quadrotors' tfs due to collision and intersections of tfs. This slows down the performance of the Gazebo simulator to represent the visualization of the results.

This is not desirable and it may even fail the mission. In order to fix this a collision detection system must be implemented which we will discuss in detail some possible options to avoid collisions. In the solution sections we will explain 3 approaches that tackle this issue in different ways that are appropriate for specific situations. After the thorough description, a comparison between them will give us some conclusions about which one is the most appropriate choice.

5.3.3 Problem 2: Moving to the closest object

So far the mission descriptions have been written in the sense that the object each quadrotor is going to search, grab and move is already predefined and chosen a priori. This means that independently of the object location and quadrotor's initial position, every quadrotor would go to the same object regardless of distance, complexity or efficiency. A generic example of the two approaches to understand the problem is shown below.

Quadrotor 1	*go to*	Object A
Quadrotor 2	*go to*	Object B
Quadrotor 3	*go to*	Object C

In the first approach each quadrotor goes to a specific object without any knowledge about the location of the object relative to the quadrotor. An alternative approach consists on each quadrotor moving to its closest object:

Quadrotor 1	*go to*	Closest Object to Quadrotor 1
Quadrotor 2	*go to*	Closest Object to Quadrotor 2
Quadrotor 3	*go to*	Closest Object to Quadrotor 3

Knowing the location of the object relative to the position of the quadrotors it is possible to calculate the distance to decide which quadrotor is more suitable to reach the object. This option is more efficient as each quadrotor will reach its closest object, decreasing the likelihood of potential collisions due to crossed paths. In a way, solving this problem may result in a solution to the first problem of section 5.3.2. Additionally, an advantages of using the second approach is that it results on a mission description that can adapt to the change of the object's location without altering the efficiency of the generated paths. This means that if Object A and Object B switch place and Object A is no longer the closest object to the Quadrotor 1, due to this description, the controller will generate a path to Quadrotor 1 that is identical to the generated path of the previous simulation where Object A was the closest object to Quadrotor 1.

5.3.4 Problem 3: grabbing same object

The problem discussed in the previous section may originate some additional problems that before weren't present in the mission. Specifying for each quadrotor to search, grab and move the object that is closest to its position may cause, in some situations, for two quadrotors to reach the same object.

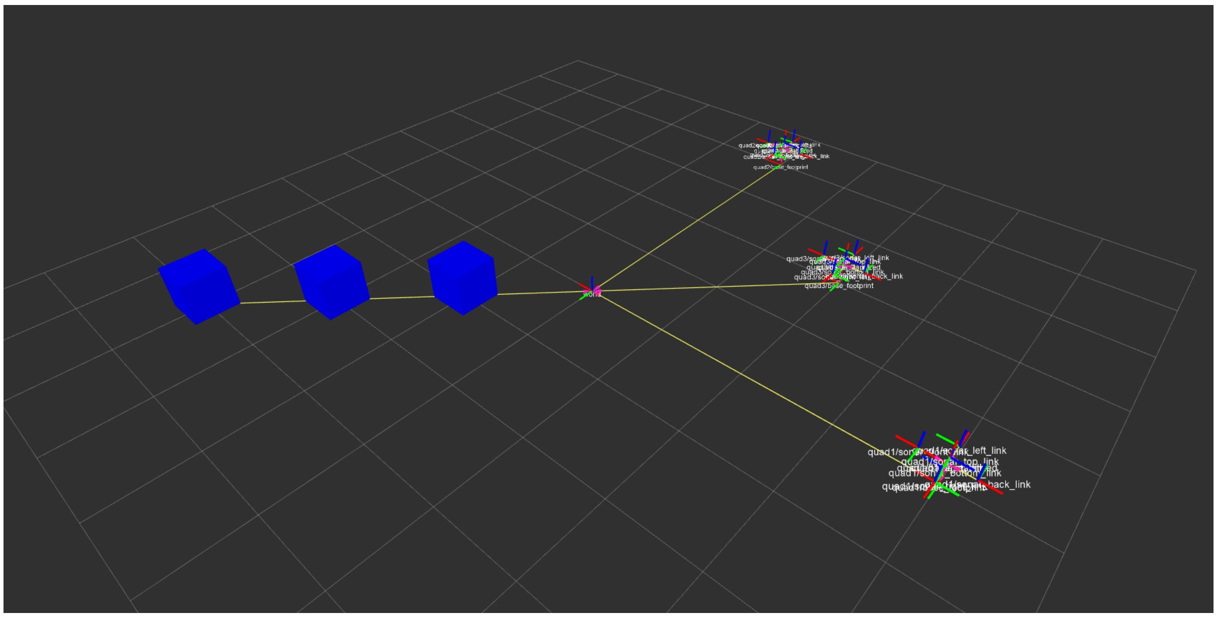


Figure 5.9: An example where the closest object causes problems

In Figure 5.9 we see an example that with the specifications described in the second approach of the section 5.3.3 all the quadrotors have the same object has their closest object. This causes a conflicting situation between the quadrotors that may even lead to unintentional collisions. For this example, the STL specifications written as described previously will not be able to fix this issue.

For these situations, there must be a signal that prevents two quadrotors from reaching the same object. Even though the same object is the closest one, the moment one vehicle decides to move in the direction of the object, the second vehicle needs to move to the second object shortest to it. This process of locking the object, preventing the other vehicles from choosing it, is to be applied to all the quadrotors and it should be of policy first-come first-serve. In case the second vehicle wants to go to the second object but the object is already taken by the third vehicle, the second vehicle should move to the third object shortest to it, in this case the last object available.

This issue and all the other problems mentioned in the sections above are going to be pinpointed and a possible solution will be discussed on every situation in the next sections. A STL solver that is able to generate a feasible controller needs to take into account all this situation since it can happen in a mission-like environment of **Challenge 3** from the MBZIRC 2017. For every section an example proving that the solution works will be shown as well as the advantages of using it in comparison with other alternatives.

5.3.5 Solution 1 to Problem 1: go up

The first solution to the issue discussed in section 5.3.2 is referred here in this thesis as ”Go Up”. This solution basically means that to avoid collision of two quadrotors whenever their position reaches a certain distance between them that is shorter than a specific threshold, one of the quadrotors, usually the one with the highest altitude, will receive an increment to the current goal location in the z coordinate. This collision avoidance will be performed by a new node called Collision Detection System (CDS). In Figure 5.10 is a representation of the additional connections added to the existing mission with 3 quadrotors.

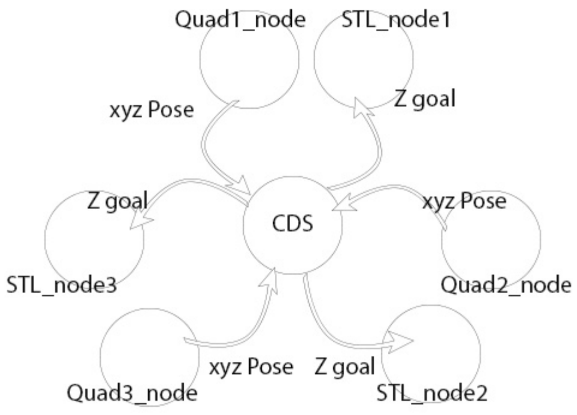


Figure 5.10: Representation of the Collision Detection System (CDS) node and its connections for the Go Up System. The CDS node subscribes to the Cartesians coordinates of all the quadrotors nodes and publishes to the z coordinate of the current goal position sent for the correspondent quadrotor # node (STL_node#).

While the simulation is undergoing, the CDS node is running simultaneously, comparing the positions of all the quadrotors, checking of any possible collisions. Whenever, the current position of two quadrotors is too close, an increment of 1 meter to the z goal coordinate is added to the quadrotor that has the highest z position to order it to ascend, avoiding any contact with the other quadrotor that continues its path without any alterations.

The example tackled in section 5.3.2 present in Figure 5.8 showed the collision of quadrotor 1 and 2, and then also, due to uncontrollable trajectory, the collision of all three quadrotors. Using this solution we can avoid the first collision by altering the goal position of quadrotor 1, ordering instead of going to goal position (0, 0, 2), the same goal as quadrotor 2, to ascend to goal position (0, 0, 4), by incrementing the current goal in the z axis until there is a safe margin for both quadrotors to perform their paths. This can be seen in Figure 5.11 where a step-by-step representation of the solution is shown.

This solution is a viable option to any type of collisions because no matter in what state each quadrotor is, whether it is moving towards the object, grabbing the object, or dropping the object, this system does not affect the outcome of the mission. For the case of grabbing the object there is no problem of missing the goal since we assume that every object has its own location, there is no overlapping of two objects. So the final goal position, which is exactly above the respective object to perform the grabbing, is never altered by the CDS node.

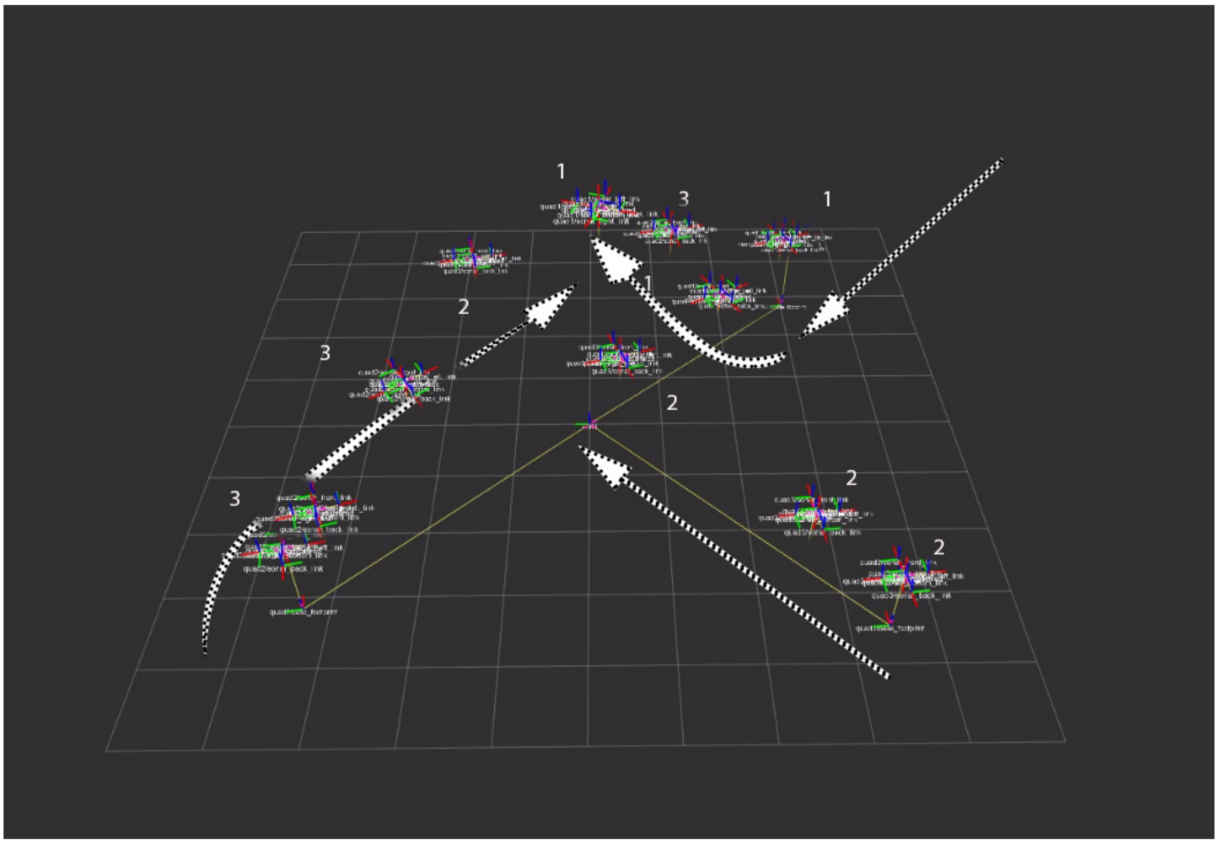


Figure 5.11: The Solution 1 in action for the mission example discussed in Section 5.3.2. In the case of the crash between quadrotor 1 and 2, now we see quadrotor 1 fly above quadrotor 2 avoiding the collision. This happens because when both quadrotors were reaching the same location $(x, y, z) = (0, 0, 2)$, before the distance between both quadrotors reaches a critical situation, an increment in the goal position z axis of quadrotor 1 is added. This results in quadrotor 1 avoiding quadrotor 2 and resuming their trajectories without any collisions whatsoever. YouTube video: `https://www.youtube.com/watch?v=nE3yL_sM4F4`

5.3.6 Solution 2 to Problem 1: Stop and Wait

We seen one possible solution to avoid collision between vehicles, however there is an alternative that doesn't have to change the path generated by the STL solver. It uses the simple rule of traffic essential for anyone when attempting to cross a street or even by a road vehicle when entering an intersection. Before attempting to cross the street it is advisable to wait and see if a moving vehicle is about to pass. If that is the case you wait until the car passes by and then you can resume your way safely. This is the same technique that is applied here called in this thesis as the ”Stop and Wait” approach.

When two quadrotors are moving and their distance is getting shorter, if they get too close to each other one of them stops their motion and lets the second quadrotor continue its path before resuming its trajectory. An representation of the new node that performs this action is shown in Figure 5.13. Whenever the distance between two vehicles is too close we replace the velocity commands from the PI controller to follow the goal points by sending a command with zero linear velocity in all direction to stop the quadrotor movement. Note that in the Gazebo simulator sending zero velocity commands in any cartesian coordinate represents a zero motion of the quadrotor in the world, this means that the velocity of the quadrotor in the world is zero in any direction. This results in a still position of the vehicle.

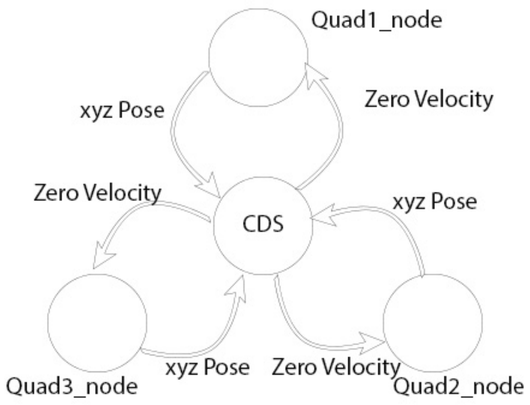


Figure 5.12: Representation of the CDS node and its connection for the Stop and Wait System. The CDS node subscribes and publishes to the same node of each quadrotor (Quad#_node). Subscribing to the same topic as in the CDS node from Section 5.3.5 and publishing to the velocity topic cmd_vel of each quadrotor.

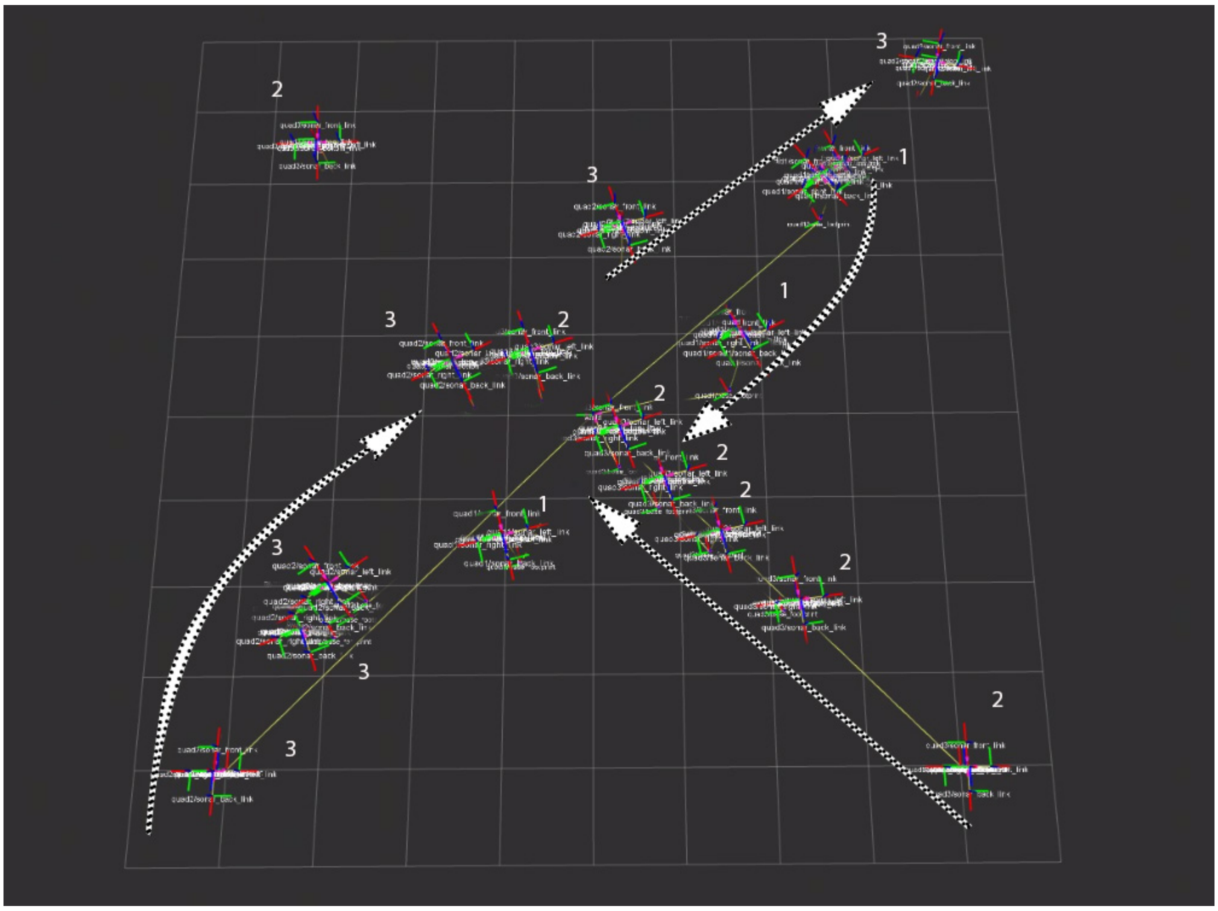


Figure 5.13: The Solution 2 in action for the mission example discussed in Section 5.3.2. In the case of the crash between quadrotor 1 and 2, when both quadrotors are reaching the same location (x, y, z) = $(0, 0, 2)$, before the distance between both quadrotors reaches a critical situation, quadrotor 1 remains still until quadrotor 2 resumes its trajectory and the distance between them reaches a save margin. This results in quadrotor 1 avoiding quadrotor 2 and resuming their trajectories without any collisions whatsoever. YouTube video: `https://www.youtube.com/watch?v=wHOPWZKqtic`

In Figure 5.13 it shows the same example of section 5.3.2 with the ”Stop and Wait” approach. We can see that at the point where quadrotor 1 and 2 cross paths, quadrotor 1 stops before reaching (0, 0, 2) goal position and waits for quadrotor 2 to reach the goal and continue its path before resuming its trajectory. This solution is a perfect feasible option for this specific example, however its not always the best alternative and may even cause problems. Let us imagine two quadrotors that in a specific point in time they both have to follow the same exact path to the object. With this approach, if a collision is imminent and the quadrotor in front is the one to stop the mission fails to be completed. The front quadrotor stops and waits for the second to continue its path, since the path is the same the collision is unavoidable. The Collision Detection System (CDS) would have to acquire prior knowledge of the quadrotors' trajectory in order to decide which vehicle to stop the movement. This would be necessary for it to solve every possible scenario. Due to the complexity of such challenge **Solution 3** was developed that can adapt to every possible configuration of the environment without compromising the mission.

Note that in the possibility for knowing in advance the path from each quadrotor, different collision avoidance approaches can be taken that best suit the problem and that assure the successfulness of the mission.

5.3.7 Solution 3 to Problem 1: STL specs

This next solution is going to take advantage of the Signal Temporal Logic synthesis to solve the collision avoidance problem. STL synthesis uses real time continuous signals so we are able to describe rules to prevent any quadrotor to get close to another quadrotor. The following specifications present a mission that involves 3 quadrotors to reach an object in the environment similar to the one defined in section 5.3.2. In this mission we also specify that it must be verified a minimum distance of 1 meter between every 2 quadrotors for the whole simulation time.

$$\square_{[0,\infty]}\sqrt{(x_1(t)-x_4(t))^2+(x_2(t)-x_5(t))^2+(x_3(t)-x_6(t))^2}>1$$
$$\square_{[0,\infty]}\sqrt{(x_4(t)-x_7(t))^2+(x_5(t)-x_8(t))^2+(x_6(t)-x_9(t))^2}>1$$
$$\square_{[0,\infty]}\sqrt{(x_7(t)-x_1(t))^2+(x_8(t)-x_2(t))^2+(x_9(t)-x_3(t))^2}>1$$

$$\Diamond_{[0,6]}(x_1(t)>-4\wedge x_1(t)<-3.5\wedge x_2(t)>3.5\wedge x_2(t)<4\wedge x_3(t)>0\wedge x_3(t)<0.5)$$
$$\Diamond_{[0,6]}(x_4(t)>3.5\wedge x_4(t)<4\wedge x_5(t)>-4\wedge x_5(t)<-3.5\wedge x_6(t)>0\wedge x_6(t)<0.5)$$
$$\Diamond_{[0,6]}(x_7(t)>3.5\wedge x_7(t)<4\wedge x_8(t)>3.5\wedge x_8(t)<4\wedge x_9(t)>0\wedge x_9(t)<0.5)$$
$$\Diamond_{[0,3]}(x_3(t)>3\wedge x_3(t)<5)\wedge(x_6(t)>3\wedge x_6(t)<5)\wedge(x_9(t)>3\wedge x_9(t)<5)$$

This is described by the first 3 lines of STL specifications where every specification start with the temporal operator $\square$, also known as 'always', and with a time interval that goes from 0 to ∞, which represents that this specifications is to hold continuously from the moment the process starts until the end of the simulation without ever being broken. The other STL specifications are to define the position of each object and the respective quadrotor that is going to grab it. Each quadrotor would have 6 time steps to reach the object location, and in those 6 time steps, in the first 3 time steps every quadrotor has to reach a certain altitude. This last line is to 'force' every quadrotor to lift off and perform their trajectory at a safely distance from the ground, since in this simulation the model used for each quadrotor is a simple integrator model with 3 degrees of freedom.

The mission with the STL specifications described above, the control input for the integrator model with a bound of 5 m/s, and the cost function of equation 4.15, the generated controller gives the solution presented in Figure 5.14. Each quadrotor has a initial position marked by t_0^i where i is the number of the

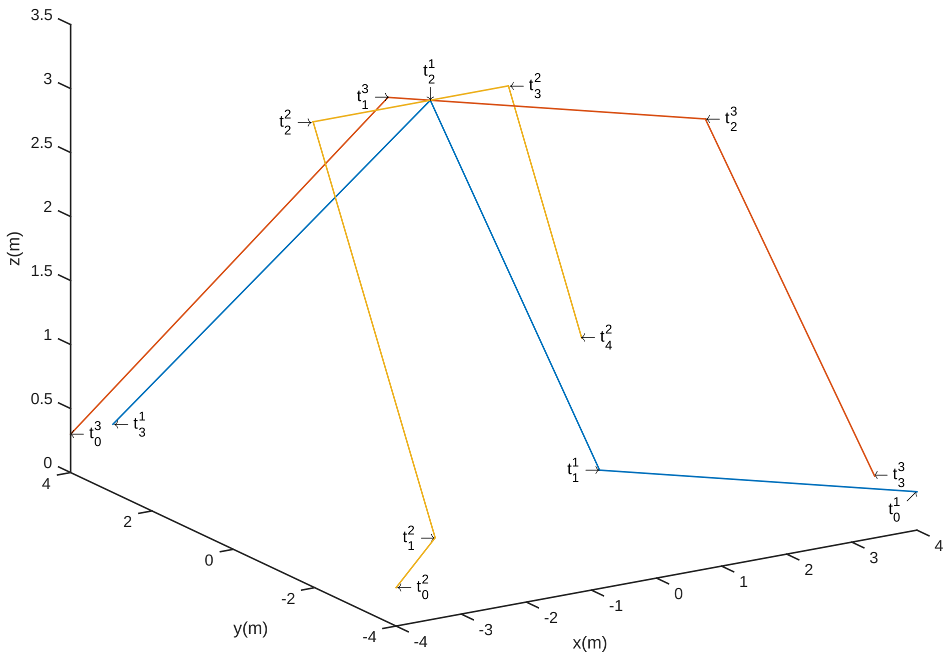


Figure 5.14: 3 quadrotors reach and grab object mission with collision avoidance activated

quadrotor. Note that the i number of the quadrotor remains consistent with the previous Figures from the alternative solutions seen above. It can be seen that every quadrotor respects its specification of reaching their object in 6 time steps, with quadrotor 2 taking the longest time t_4^2. It can be seen relatively easily that at every time step of full duration of the mission the distance between every quadrotor is kept. Even the point where the trajectories of the quadrotor cross (t_2^1), every quadrotor crosses that point at different time step of the simulation, with only quadrotor 1 having it as a 'goal point', the remaining ones the point is just the intersection between two 'goal points'. Note that this solution, even though in simulation the collision avoidance is working throughout the duration of the mission, the same can't be always guaranteed when running the process in Gazbo using quadrotor models. This is due to independent processes for each quadrotor running in the same environment where the computation processing can slow down one or two quadrotors and so the time steps for each quadrotor may not correspond to the ones in the simulation. To prevent such cases it is recommended to add one of the alternative solutions discussed in the previous section just as a precaution measure.

Given this 3 options we can conclude that this are all valid solutions and each can be used to solve specific missions. Solutions 1 and 2 are of the same kind and are applied to solve collisions in the

Gazebo environment where the path generated has no collision avoidance system. For Solution 3, STL specifications are introduced to solve any possible collision that may happen during the generation of the quadrotors' trajectories however, when running in the Gazebo environment it may be advisable to add one of the previous solutions just to insure that no disturbance cause unpredictable crashes. This applies to only simulated scenarios.

5.3.8 Solution to Problem 2: STL specs

In this section we are going to address the issue from section 5.3.3. The objective is to generalize the mission so that every quadrotor would grab its closest object. With this, the location of each object could change and the mission would adapt each quadrotor to grab the proper object generating trajectories that most effectively satisfy the specifications. Grabbing the closest object allows for shorter paths to the object as well as less crossed trajectories which in return increases the likelihood of success in the mission. In order to grab the closest object, it is necessary to compare distances between the current quadrotor and the three locations of the objects. This is achieved by the first equation presented below.

$$\begin{aligned}
&\Diamond_{[0,3]}(x_{i_x}(t) - Obj_{1_x}(t))^2 + (x_{i_y}(t) - Obj_{1_y})^2 < (x_{i_x}(t) - Obj_{2_x}(t))^2 + (x_{i_y}(t) - Obj_{2_y})^2 \wedge \\
&\qquad (x_{i_x}(t) - Obj_{1_x}(t))^2 + (x_{i_y}(t) - Obj_{1_y}(t))^2 < (x_{i_x}(t) - Obj_{3_x}(t))^2 + (x_{i_y}(t) - Obj_{3_y}t))^2 \;\; => \\
&\Diamond_{[0,3]}(x_{i_x}(t) > Obj_{1_x}(t) \wedge x_{i_x}(t) < Obj_{1_x}(t) + 0.5 \wedge \\
&\qquad x_{i_y}(t) > Obj_{1_y}(t) \wedge x_{i_y}(t) < Obj_{1_y}(t) + 0.5 \;\wedge x_{i_z}(t) > 0 \wedge x_{i_z}(t) < 0.5) \\
&\Diamond_{[0,3]}(x_{i_z}(t) > 3 \wedge x_{i_z}(t) < 5)
\end{aligned}$$

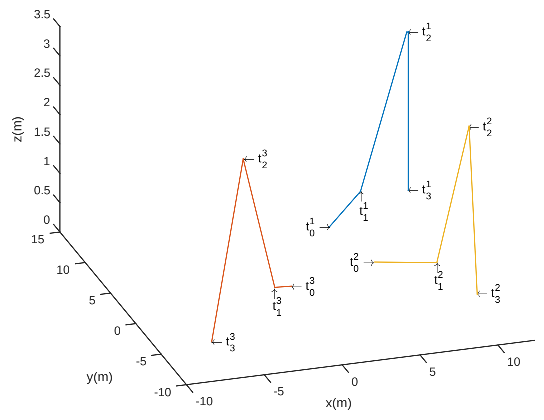
For each quadrotor i = {1, 2, 3} we calculate, at every time step, the distance between the quadrotor and the objects in the environment, and one way to do this is by conditional syntactics. If the distance between quadrotor i and Obj_1 is smaller than the distances to either of the remaining objects then the obvious choice is to proceed to move towards Obj_1. This is exactly what is exemplified by the first equation, by comparing the distances between the quadrotor and all the objects, if we prove that they are smaller we proceed, hence the boolean symbol next (=>), by going to the object that is the closest. This STL specification needs to be defined for 3 different possibilities of outcome, and then the rest is the same as it was written in the previous examples with the modification of each trajectory generated being defined only when one of the conditional syntax is valid.

In Figure 5.15 we have 3 different examples using the STL specifications explained above. For these examples, it was used the same characteristic of the STL solver as in section 5.3.8 for bound, cost function, etc. However, different location of the object were chosen to illustrate the solution developed in this section, which can be seen in Table 5.1, as well as some computational time averages to study the complexity of the mission. The conclusion taken from simulating different examples using this STL specifications is that is it very computational demanding for the CPU to perform this continuous calculations of all the distances and comparing to find the closest object. Nevertheless, every quadrotor goes to the closest object and due to the STL syntax of describing continuous real time signal it is possible to move each quadrotor to the closest dynamical object, as long as the trajectory of the object is fully

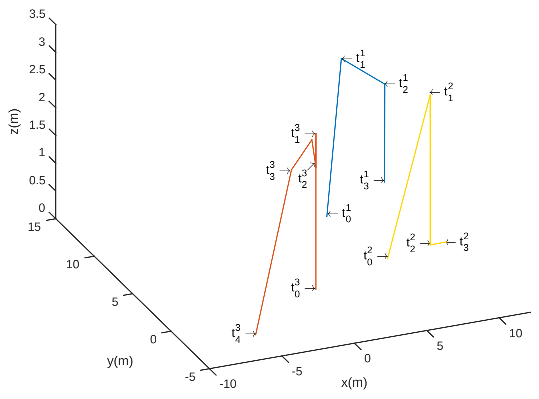
Example	Quadrotor 1 t_0 (x, y, z)	Quadrotor 2 t_0 (x, y, z)	Quadrotor 3 t_0 (x, y, z)	Obj_1 (x, y)	Obj_2 (x, y)	Obj_3 (x, y)	time (sec)
5.15(a)	(5, 8, 0.3)	(0, 0, 0.3)	(6, 2, 0.3)	(11, 11)	(12, -8)	(-8, -8)	293
5.15(b)	(5, 8, 0.3)	(0, 0, 0.3)	(6, 2, 0.3)	(10, 10)	(10, 2)	(-7, -5)	374
5.15(c)	(5, 8, 0.3)	(0, 0, 0.3)	(6, 2, 0.3)	(10, 5)	(-5, 5)	(6, -6)	412

Table 5.1: Characteristics of the different examples of Figure 5.15

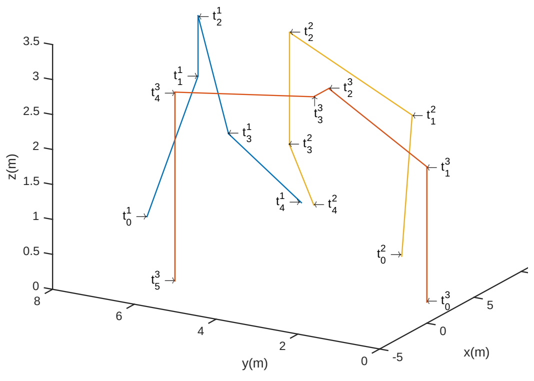
known before the mission is initiated.



(a) Mission 1: each quadrotor has the first object closest to it has the closest object.



(b) Mission 2: quadrotor 1 and 2 have 2 objects relatively close to each other.



(c) Mission 3: quadrotor 1 and 2 have 1 object to be the closest object to both quadrotors. This is a problem that is going to be discussed in the next section.

Figure 5.15: Different missions involving 3 quadrotors reaching 3 distinguished objects in which the locations of the objects are changed. The goal is to prove that with the STL specifications described in this section each quadrotor will go to its closest object in all different cases.

However, with the use of this specifications another problem has been originated, which was verified by the example performed and illustrated in Figure 5.15(c). If there is the case of the closest object to two distinguished quadrotors be exactly the same, the result is an undesirable behavior. This will cause two quadrotors to grab the same object, and in the event of adding collision avoidance systems from the previous section would only add to the unsuccessfulness of the mission. This situation should not happen in any circumstances and so in order to avoid those occasions we developed a solution that prevents those conflicts appearing. This is going to be addressed in the next section which solves this last problem that may occur in our mission.

The solution given by this STL specifications gives us a feasible solution to reach the closest object for every quadrotor, and despite the long time to generate a controller for the STL solver, it provides a more generic approach to the **Challenge 3** mission by giving us feasible trajectories with few crossing points than normal.

5.3.9 Solution to Problem 3: + STL specs

This final section involving solutions for the problems discussed before we are going to solve the problem caused by wanting the quadrotors to grab the closest object to its position. Even though it allowed us to generated much simpler trajectories for all the quadrotors, in the cases of having the same object close to two distinguished quadrotors the problem shown in Figure 5.9 of section 5.3.4 would generate an output like the one represented in Figure 5.15(c). In order to solve this problem we developed alternative STL specifications as the ones described below.

$$
\begin{aligned}
&\Diamond_{[0,3]}(x_{1_x}(t) - Obj_{1_x}(t))^2 + (x_{1_y}(t) - Obj_{1_y})^2 < (x_{1_x}(t) - Obj_{2_x}(t))^2 + (x_{1_y}(t) - Obj_{2_y})^2 \wedge \\
&\qquad (x_{1_x}(t) - Obj_{1_x}(t))^2 + (x_{1_y}(t) - Obj_{1_y}(t))^2 < (x_{1_x}(t) - Obj_{3_x}(t))^2 + (x_{1_y}(t) - Obj_{3_y}t))^2 \;=> \\
&\qquad \Diamond_{[0,3]}(x_{1_x}(t) \approx Obj_{1_x}(t) \wedge x_{1_y}(t) \approx Obj_{1_y}(t) \wedge x_{1_z}(t) \approx 0) \\
&(x_{1_x}(t) \approx Obj_{1_x}(t) \wedge x_{1_y}(t) \approx Obj_{1_y}(t) \wedge x_{1_z}(t) \approx 0) \wedge \\
&\qquad (x_{2_x}(t) - Obj_{2_x}(t))^2 + (x_{2_y}(t) - Obj_{2_y})^2 < (x_{2_x}(t) - Obj_{3_x}(t))^2 + (x_{2_y}(t) - Obj_{3_y})^2 => \\
&\qquad \Diamond_{[0,3]}(x_{2_x}(t) \approx Obj_{2_x}(t) \wedge x_{2_y}(t) \approx Obj_{2_y}(t) \wedge x_{2_z}(t) \approx 0) \\
&(x_{2_x}(t) \approx Obj_{2_x}(t) \wedge x_{2_y}(t) \approx Obj_{2_y}(t) \wedge x_{2_z}(t) \approx 0) => \\
&\qquad \Diamond_{[0,3]}(x_{3_x}(t) \approx Obj_{3_x}(t) \wedge x_{3_y}(t) \approx Obj_{3_y}(t) \wedge x_{3_z}(t) \approx 0) \\
&\Diamond_{[0,3]}(x_{i_z}(t) > 3 \wedge x_{i_z}(t) < 5)
\end{aligned}
$$

The specifications are somewhat similar to the ones described in the previous section. The first line is completely identical to the previous solution since the first quadrotor has all the objects available at his disposal. After the first quadrotor determines the closest object to its position, the calculation for the second quadrotor are done differently. As it can be seen in the second line of the specifications it is checked if the first quadrotor has reached the object determined as the closest to its position. Only after that is verified will the calculations to determine the closest object to the second quadrotor will proceed. Notice that a comparison is only between the two remaining since the first object was checked as true for the first quadrotor. The closest object to the second quadrotor will be smaller then the other remaining object, so we proceed, hence the boolean symbol next ($=>$), to the move in the direction of the closest object. Once the second quadrotor reaches the object the third line of specifications describes the movement of the third quadrotor to the remaining object. The last line is identical to the previous solution in the sense of ordering quadrotor $i = \{1, 2, 3\}$ to go upwards as soon as the simulation starts so it won't collide with the floor when navigating.

The outcome of this solution, which can be visualized in Figure 5.16 is quite as we expected. The objects are in position (5,5,0), (6,6,0) and (7,7,0), and with the STL specifications described in this example every quadrotor goes to a different object that is its closest object that hasn't been taken by any of the other. There is some drawbacks from this alternative since the method of choice is based on first-come first-serve which may not be the best alternative for every case but it is not our goal to achieve perfect efficiency. For this thesis the objective is to find an alternative in terms of feasibility of the mission, where there is no issues that may cause it to not complete the planned task.

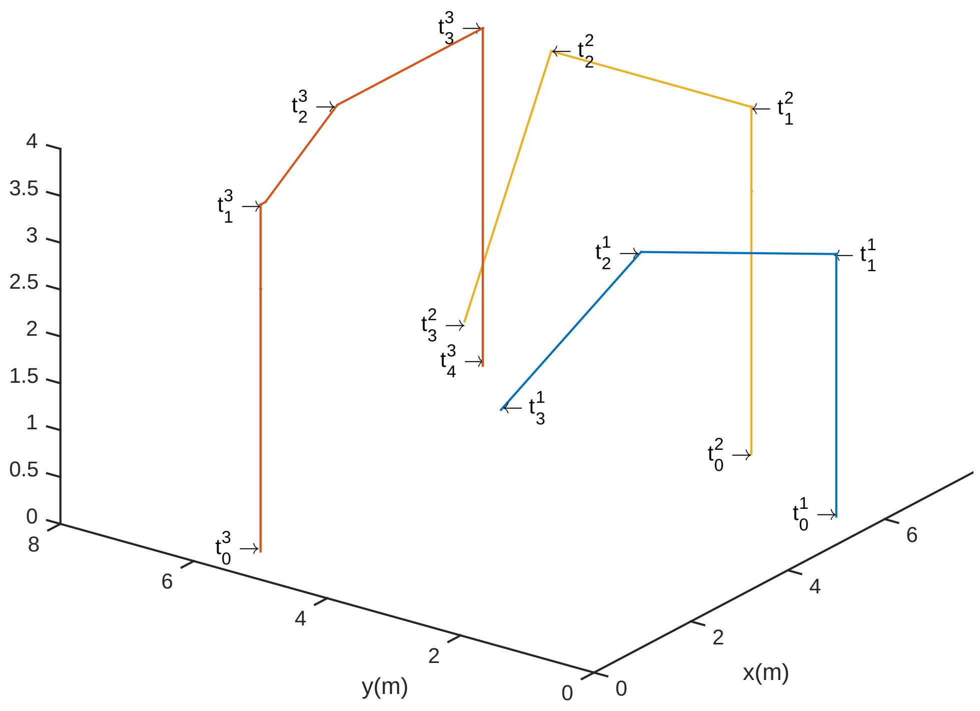


Figure 5.16: An example of a mission with 3 quadrotors going to the closest object that is not taken by any other previous quadrotor

5.4 Final solution

After all problems have been addressed it is time to add everything together with the purpose of reproducing the **Challenge 3** of the international competition MBZIRC 2017. The mission, as it was described previously, involves a coordinated mission between 3 quadrotors to search and grab 3 individual objects and one object cooperatively in a dynamical environment. The first part consists of the 3 quadrotors starting from a known location and searching the objects in the environment. The high level controller developed in this thesis allows us to focus primarily on the trajectory generated of each model leaving the more low-end priorities such as locating the objects and its physical characteristics for the low level controllers. As such, we assume a priori that the exact location of each object is given to us before starting the mission. The starting point of the mission can be seen in Figure 5.17(a) identical to the one visualized in section 5.3 Figure 5.7. The first mission is completed when each quadrotor reaches its object and grabs it (Figure 5.17(b)). The second objective, performed in Figure 5.17(c), is to relocate the objects to a new position and, when the final destination of the object has been reached, the quadrotor can drop the object. This summarizes the first part of the mission we set out to do, where all the different problems discussed in the previous section can occur and where the alternative solutions can provide a feasible option that avoids any possible failing situations. The next and final part is allocated for the cooperative mission involving 2 quadrotors grasping and transporting of a fairly large size object (200$\times$20$\times$20 cm). This object respects the dimensions provided by the MBZIRC 2017 competition guidelines. The object will be grabbed by one quadrotor on each end to avoid any unpredictable collisions.

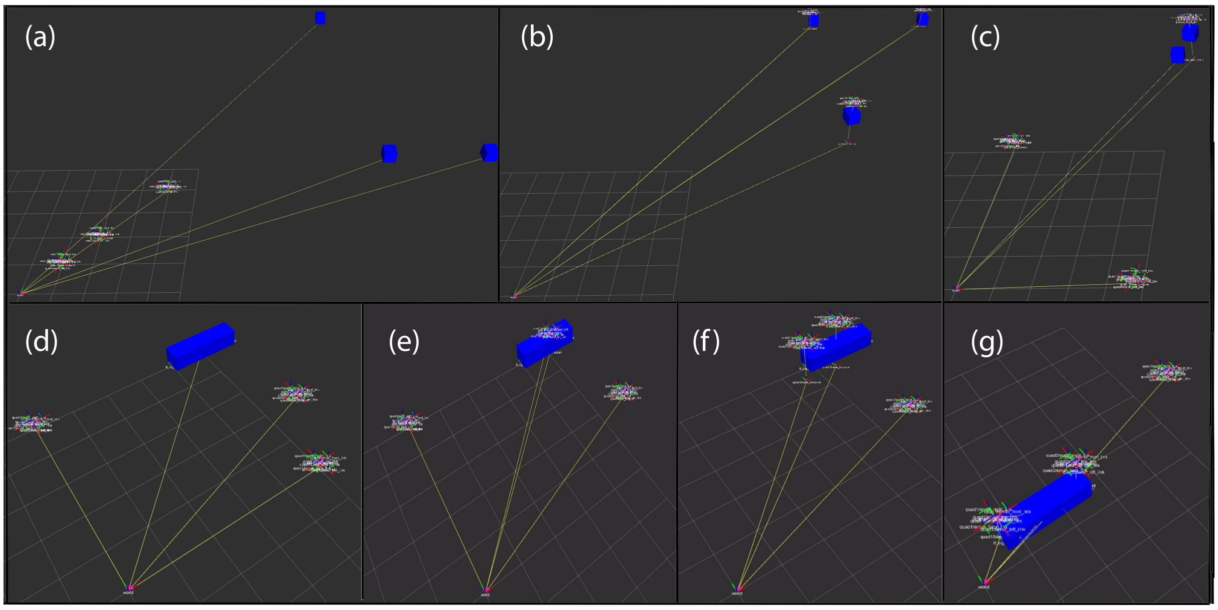


Figure 5.17: The final mission illustrated step-by-step representing every major event. In (a) is the starting point of the 3 quadrotors with the initial position of the 3 individual objects; (b) shows the quadrotors reaching the objects and transporting them to a new location; (c) illustrates the new location for all 3 objects and the new starting points of quadrotor 1 and 2; (d) is where the new mission starts with the big object initialized, and quadrotor 3 is shown in its final position where it will stay for the remaining of the time; (e), (f), and (g) are the 3 step process of moving the big object: quadrotor 1 goes to the object to evaluate its size, then quadrotor 1 waits for quadrotor 2 to arrive since it can't transported by itself, and finally both quadrotors moving the object to its final location. YouTube video: `https://www.youtube.com/watch?v=Y-KO_NPfYvw`

This cooperative team work is achieved by expressing the desired movement of the two quadrotors present in this objective. In a cooperative mission two or more vehicles are working together to fulfill a goal, in our case is to grab an object that can't be transported by just one quadrotor. Knowing that the current environment contains both types of objects (individual objects and big objects), the quadrotors need to be able to distinguish between the two in order to perform the right mission. For the scope of this thesis we don't address properties of the object such as weight, type of material, etc. We assume that the quadrotor identifies automatically what object is dealing with in contact. The mission describes then a first quadrotor reaching its closest object, identifies that it is in fact a big object, and waits for the second quadrotor to come to its assistance. Note that the STL specifications written next are simplified to just one possible case. This is the case when Object 1 is closer than Object 2 or 3. However, this may not always be the case as such additional STL specifications are added to deal with all possible scenarios.

$$\begin{aligned}
&\Diamond_{[0,3]}(x_{1_x}(t) - Obj_{1_x}(t))^2 + (x_{1_y}(t) - Obj_{1_y})^2 < (x_{1_x}(t) - Obj_{2_x}(t))^2 + (x_{1_y}(t) - Obj_{2_y})^2 \wedge \\
&\quad (x_{1_x}(t) - Obj_{1_x}(t))^2 + (x_{1_y}(t) - Obj_{1_y}(t))^2 < (x_{1_x}(t) - Obj_{3_x}(t))^2 + (x_{1_y}(t) - Obj_{3_y}t))^2 \;\; => \\
&\quad \Diamond_{[0,3]}(x_{1_x}(t) \approx Obj_{1_x}(t) \wedge x_{1_y}(t) \approx Obj_{1_y}(t) \wedge x_{1_z}(t) \approx 0) \\
&(x_{1_x}(t) \approx Obj_{1_x}(t) \wedge x_{1_y}(t) \approx Obj_{1_y}(t) \wedge x_{1_z}(t) \approx 0) \;\; => \Box_{[4,\infty]}(x_{1_x} + 1 < x_{2_x}(t) < x_{1_x} + 2 \;\wedge \\
&\quad x_{1_y} < x_{2_y}(t) < x_{1_y} + 0.2 \wedge x_{1_z} < x_{2_z}(t) < x_{1_z} + 0.2) \;\wedge \\
&\quad \Diamond_{[4,8]}(x_{1_x}(t) \approx Goal_x(t) \wedge x_{1_y}(t) \approx Goal_y(t) \wedge x_{1_z}(t) \approx Goal_z(t)) \\
&\Diamond_{[0,3]}(x_{i_z}(t) > 3 \wedge x_{i_z}(t) < 5)
\end{aligned}$$

The solution to the problem described above is represented in Figure 5.18. The first part of the equations describes simply the first quadrotor reaching its closest object. The same identical STL specification described in all missions (section 5.3.9) involving quadrotors attempting to grasp the closest individual object to its position. The second equation is what allows the mission to adapt to the situation and call for backup when the object requires a second quadrotor to be lifted. Again, we assume the moment

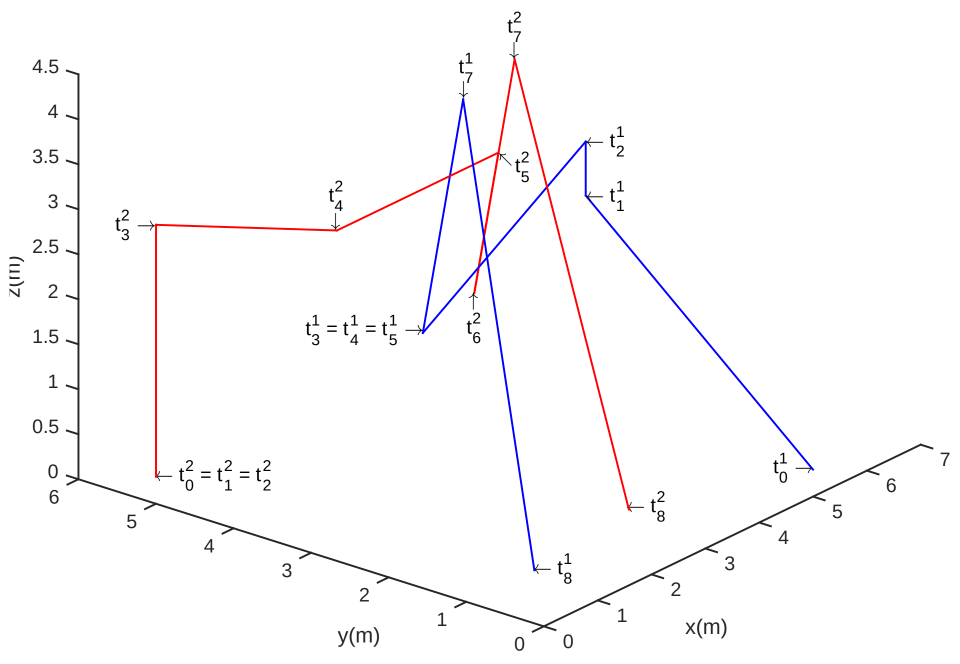


Figure 5.18: The trajectories path of the quadrotors involved in the cooperative mission of transporting a big object.

quadrotor 1 reaches the object (t^1_3) it has on-board sensors that allow to distinguish the current object from the ordinary single object as well as send a signal to quadrotor 2 that it requires its assistance. After those low-level conditions have been established the second part of the equation can be used. This equation will only hold true when the quadrotor 1 is on top of the object. After that the STL specifications applied to quadrotor 2 and the cooperative mission will be in play. Note that quadrotor 2 is in its initial position from time t = 0 to t = 2, this is due to quadrotor 1 not having reached the object at those time steps. When at time t = 3 quadrotor 1 reaches the object then the cooperative mission starts, as the second equation holds true and orders are given. This specifications are to the right of the boolean operator (=>) to represent the next steps to make after quadrotor 1 reaches the object. This orders quadrotor 2 to converge its position to the location of quadrotor 1 (t^2 reach t^1_3), at the same time keep a safe distance to it and also respect the object dimensions so it won't miss the grasping. This is the reason why t$^1_3$ and t$^2_6$ are not the exact same location, due to the dimension of the object defined previously. The last part of the STL specification is regarding the new goal position of the object to be dropped. Since the previous rule needs to be kept from t = 4 until the end of the simulation ($\square_{[4,\infty]}(x_{1_x} + 1 < x_{2_x}(t) < x_{1_x} + 2 \wedge x_{1_y} < x_{2_y}(t) < x_{1_y} + 0.2 \wedge x_{1_z} < x_{2_z}(t) < x_{1_z} + 0.2)$) by ordering quadrotor 1 to move to that new goal, quadrotor 2 will automatically follow the same path creating the so desired cooperative mission between two quadrotors transporting a big object as seen from Figure 5.17(f) until it is dropped viewed in sub-picture (g). However to replicate this same parallel motion into the Gazebo simulator, some changes needed to be done to the way data is sent and processed between nodes. As a result, from the moment the two quadrotors reach the object and the object is grabbed the goal positions from the trajectory generated by the STL solver will be sent at the same time synchronized by each STLnode$\#$ of each quadrotor. This prevents one of the quadrotors to move faster than the other one, making it a smooth transportation of the object without any disturbances.

Chapter 6

Conclusions

In this thesis we present an option to solve a very practical problem of a multi-vehicle environment using STL specifications. This mission involved 3 quadrotors to search, grab and transport objects from one position to another, with occasional cooperative movement of one object by two quadrotors. This required coordination from all the vehicles so that the same object would not be grabbed and also unintentional collisions could occur that may fail the mission.

6.1 Achievements

The major achievements of the present work can be summarized to being able to show the advantages of using Signal Temporal Logic language to describe the mission specifications on a high-end level controller. Previous research papers have been written on using different techniques with LTL language that prove the feasibility of multi-vehicle coordinated missions [19] [22]. In this thesis we present the differences between the existing work using LTL with the now new alternative language that is STL. The advantageous being that its variables are continuous-time signals rather than discrete-time signals allowing for a more robust satisfaction parameter than the previous yes or no answer for the satisfiability in discrete-time. Additionally, the use of a MPC controller that allows us to adapt in real time to the current situation to disturbances that may happen ahead of time, giving us results less prone to feasibility problems than the ones given by previously researched methods.

6.2 Future Work

Even though the major goals of this thesis were achieved, it must be noted that this is still a very particular example and it is only fully functional in a completely known environment with perfect conditions. It is true that the goal of this thesis was to solve the problem **Challenge 3** from the international competition MBZIRC 2017. Nevertheless, it would be interesting to adapt our STL specifications to a mission with more than 3 quadrotors. As some of the missions developed in this thesis are limited to only 3 vehicles. Additionally, it would be interesting to understand how to deal with unknown or partially known object locations as well as uncertainties in the quadrotor current location.

Although a complete controller was developed and simulated it was not possible to run the tests with the real system. Due to the lack of knowledge on the real hardware and not enough time to implement my own external algorithm, it was only possible to test the controller.

The last thing to be noted is that, despite only being tested for environments adapted to the **Challenge 3** problem, the contribution of this thesis can be used as a development environment and testbed for future work.

Bibliography

[1] Astrom, K. J. (2002). Pid control. In *Control System Design*, pages 216–252.

[2] Baier, C. and Katoen, J.-P. (2008). *Principles of Model Checking*. MIT press. ISBN 978-0-262-02649-9.

[3] Bellingham, J., Tillerson, M., Alighanbari, M., and How, J. (2002). Cooperative path planning for multiple uavs in dynamic and uncertain environments. *41st IEEE Conference on Decision and Control Las Vegas NV*, 3:2816 – 2822. doi:10.1109/CDC.2002.1184270.

[4] Bemporad, A. and Morari, M. (1999). Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35:347–536.

[5] Blakelock, J. H. (1991). Longitudinal dynamics. *Automatic Control of Aircraft and Missiles*.

[6] Carreira, T. G. (2013). Quadcopter automatic landing on a docking station. *Conference on Decision and Control CDC*.

[7] Cavalier, T. M., Pardalos, P. M., and Soyster, A. L. (1990). Modeling and integer programming techniques applied to propositional calculus. *Computers and Operations Research*, pages 561–570.

[8] Clarke, E. M., Grumberg, O., and Peled, D. (1999). *Model Checking*. MIT Press Cambridge MA USA.

[9] Cook, D. and Buja, A. (1997). Manual controls for high-dimensional data projections. *Journal of Computational and Graphical Statistics*, pages 464–480.

[10] Emerson, E. A. (1995). *Handbook of theoretical computer science*. MIT Press Cambridge MA USA.

[11] Faied, M., Mostafa, A., and Girard, A. (2009). Dynamic optimal control of multiple depot vehicle routing problem with metric temporal logic. *American Control Conference Hyatt Regency Riverfront St. Louis MO USA*.

[12] Fusaoka, A., Seki, H., and Takahashi, K. (1983). A description and reasoning of plant controllers in temporal logic. *18th International Joint Conference on Artificial intelligence*, 1:405–408.

[13] Gabbay, D., Reynolds, M. A., and Finger, M. (2001). Temporal logic: Mathematical foundations and computational aspects. *Journal of Logic, Language and Information*, 10(3):406–410. doi:10.1023/A:1011212908144.

[14] He, K. (2015). Robot manipulation planning under linear temporal logic specifications. Master's thesis, Rice University.

[15] Heidarsson, H. I. (2014). Simulation and implementation of temporal logic-based motion planning for autonomous vehicles. Master's thesis, KTH Royal Institute of Technology.

[16] Hoffmann, G. M., Huang, H., Waslander, S. L., and Tomlin, C. J. (2007). Quadrotor helicopter flight dynamics and control: Theory and experiment. *American Institute of Aeronautics and Astronautics*.

[17] Jiřinec, T. (2011). Stabilization and control of unmanned quadcopter. Master's thesis, Czech Technical University in Prague.

[18] Karaman, S. (2009). Optimal planning with temporal logic specifications. MIT.

[19] Karaman, S. and Frazzoli, E. (2011). Linear temporal logic vehicle routing with applications to multi-uav mission planning. *International Journal of Robust and Nonlinear Control*, 21(12):1372?1395. doi: 10.1002/rnc.1715.

[20] Koymans, R. (1983). Specifying real-time properties with metric temporal logic. *Journal Real-Time Systems*, 2(4):255–299.

[21] Kuhlmann, K. (2014). Ltl specifications for highway lane changing maneuvers of highly automated vehicles. Master's thesis, Delft University Technology.

[22] Lacerda, B. and Lima, P. (2011). Ltl-based decentralized supervisory control of multi-robot tasks modelled as petri nets. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[23] Lebedev, A. (2013). Design and implementation of a 6dof control system for an autonomous quadrocopter. Master's thesis, Lulea University of Technology.

[24] Mahony, R., Kumar, V., and Corke, P. (2012). Multirotor aerial vehicle. *IEEE Robotics and Automation Magazine*. doi: 10.1109/MRA.2012.2206474.

[25] Monzon, I. (2013). Developing a ros enabled full autonomous quadrotor. Master's thesis, Lulea University of Technology.

[26] Ostroff, J. S. (1989). Temporal logic for real time systems. *John Wiley Sons Inc. New York NY USA*, page 209. ISBN:0-471-92402-4.

[27] Ouaknine, J. and Worrell, J. (2008). Some recent results in metric temporal logic. *6th international conference on Formal Modeling and Analysis of Timed Systems*, pages 1 – 13. doi:10.1007/978-3-540-85778-51.

[28] Raman, V., Donzé, A., Maasoumy, M., Murray, R., Sangiovanni-Vincentelli, A., and Seshia, S. (2014). Model predictive control with signal temporal logic specifications. *53rd IEEE Conference on Decision and Control CDC Los Angeles CA USA*, pages 81–87. doi:10.1109/CDC.2014.7039363.

[29] Raman, V., Donzé, A., Sadigh, D., Murray, R., and Seshia, S. (2015). Reactive synthesis from signal temporal logic specifications. *Hybrid Systems: Computation and Control HSCC Seattle WA USA*, pages 239–248. doi:10.1145/2728606.2728628.

[30] Richards, A., Bellingham, J., Tillerson, M., and How, J. (2002). Coordination and control of multiple uavs. *AIAA Guidance Navigation and Control Conference*.

[31] Rizqi, A. A. A. (2014). Path planning and formation control via potential function for uav quadrotor. Master's thesis, Gadjah Mada University.

[32] Roberts, M., Apker, T., Johnson, B., Auslander, B., Wellman, B., and Aha, D. W. (2014). Coordinating robot teams for disaster relief. *28th International Florida Artificial Intelligence Research Society Conference*.

[33] Shen, Y., Xu, D., Tan, M., and Jiang, Z.-M. (2007). Visual guided approach-to-grasp for humanoid robots. In *Humanoid Robots: Human-like Machines*, pages 642–665.

[34] Strejcek, J. (2014). *Linear Temporal Logic: Expressiveness and Model Checking*. PhD thesis, Masaryk University Brno.

[35] Ulusoy, A., Smith, S. L., Ding, X. C., Belta, C., and Rus, D. (2011). Optimal multi-robot path planning with temporal logic constraints. *6th international conference on Formal Modeling and Analysis of Timed Systems*.

[36] Vejpustek, T. (2013). Robustness analysis of extended signal temporal logic stl*. Master's thesis, Masaryk University Brno.

[37] Wolff, E. M., Topcu, U., and Murray, R. M. (2014). Optimization-based trajectory generation with linear temporal logic specifications. *Robotics and Automation ICRA 2014 IEEE International Conference on*, pages 5319–5325. doi:10.1109/ICRA.2014.6907641.

Appendix A

TF’s of 1 Quad 1 Object Example

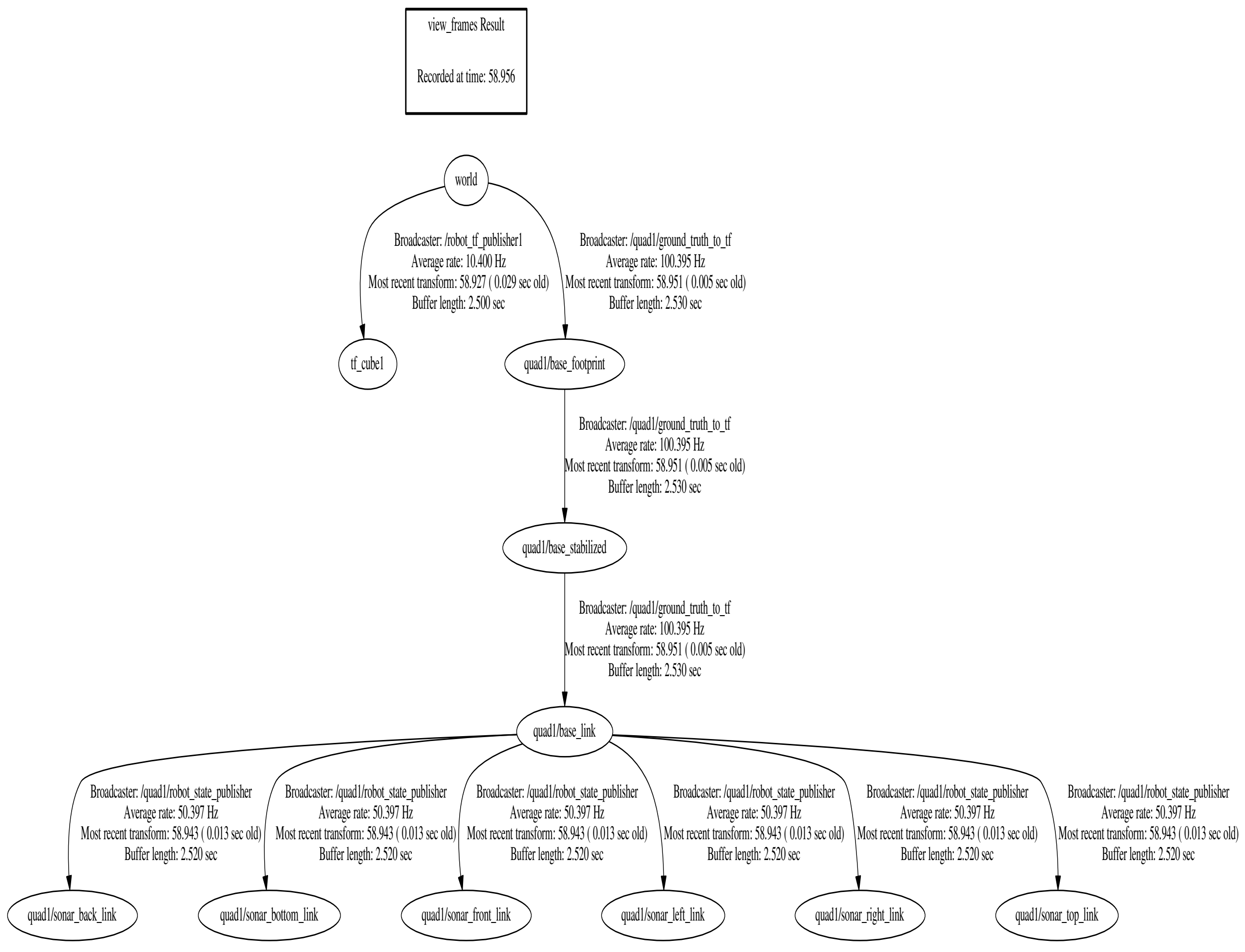

view_frames Result
Recorded at time: 58.956
world
Broadcaster: /robot_tf_publisher1
Average rate: 10.400 Hz
Most recent transform: 58.927 (0.029 sec old)
Buffer length: 2.500 sec
Broadcaster: /quad1/ground_truth_to_tf
Average rate: 100.395 Hz
Most recent transform: 58.951 (0.005 sec old)
Buffer length: 2.530 sec
tf_cube1
quad1/base_footprint
Broadcaster: /quad1/ground_truth_to_tf
Average rate: 100.395 Hz
Most recent transform: 58.951 (0.005 sec old)
Buffer length: 2.530 sec
quad1/base_stabilized
Broadcaster: /quad1/ground_truth_to_tf
Average rate: 100.395 Hz
Most recent transform: 58.951 (0.005 sec old)
Buffer length: 2.530 sec
quad1/base_link
Broadcaster: /quad1/robot_state_publisher
Average rate: 50.397 Hz
Most recent transform: 58.943 (0.013 sec old)
Buffer length: 2.520 sec
Broadcaster: /quad1/robot_state_publisher
Average rate: 50.397 Hz
Most recent transform: 58.943 (0.013 sec old)
Buffer length: 2.520 sec
Broadcaster: /quad1/robot_state_publisher
Average rate: 50.397 Hz
Most recent transform: 58.943 (0.013 sec old)
Buffer length: 2.520 sec
Broadcaster: /quad1/robot_state_publisher
Average rate: 50.397 Hz
Most recent transform: 58.943 (0.013 sec old)
Buffer length: 2.520 sec
Broadcaster: /quad1/robot_state_publisher
Average rate: 50.397 Hz
Most recent transform: 58.943 (0.013 sec old)
Buffer length: 2.520 sec
Broadcaster: /quad1/robot_state_publisher
Average rate: 50.397 Hz
Most recent transform: 58.943 (0.013 sec old)
Buffer length: 2.520 sec
quad1/sonar_back_link
quad1/sonar_bottom_link
quad1/sonar_front_link
quad1/sonar_left_link
quad1/sonar_right_link
quad1/sonar_top_link

Figure A.1: Representation of the connections of TF’s in the example with 1 Quad and 1 static Object

Appendix B

Proof of STL Controller Synthesis

This appendix contains the full detail explanation of the encoding as a MILP of the STL specifications. This encoding consists of system constraints and STL constraints [28].

B.1 System Constraints

B.1.1 Propositional calculus and linear integer programming

Consider X_i as a $literal$ which can take values of "T" (true) or "F" (false). With this $literals$ we use boolean operators such as $\wedge$ (and), $\vee$ (or), $\neg$ (not), $\Rightarrow$ (implies), $\Leftrightarrow$, (if and only if), $\oplus$, (exclusive or), to write statements like the one represented below.

$$(X_1 \vee X_2) \wedge X_3 \Rightarrow X_4 \Leftrightarrow \neg(X_5 \oplus X_6)$$

Correspondingly one can associate with a literal X_i a *logical variable* $\delta_i \in \{0,1\}$, which has a value of either 1 if X_i = T, or 0 otherwise. In [7] we get that integer programming was considered an efficient way to perform automated deduction. By converting propositional logic problems into linear inequalities involving logical variables δ_i by means of a linear integer program. Here are some examples given that can prove that equivalence.

$$X_1 \vee X_2 \text{ is equivalent to } \delta_1 + \delta_2 \geq 1 \tag{B.1a}$$

$$X_1 \wedge X_2 \text{ is equivalent to } \delta_1 = 1, \delta_2 = 1 \tag{B.1b}$$

$$\neg X_1 \text{ is equivalent to } \delta_1 = 0 \tag{B.1c}$$

$$X_1 \Rightarrow X_2 \text{ is equivalent to } \delta_1 - \delta_2 \leq 0 \tag{B.1d}$$

$$X_1 \Leftrightarrow X_2 \text{ is equivalent to } \delta_1 - \delta_2 = 0 \tag{B.1e}$$

As we are interested in systems which have both logic and dynamics, we wish to establish a link between the two worlds. For this purpose, we end up with *mixed-integer linear inequalities*, i.e linear inequalities involving both *continuous variables* $x \in \mathbb{R}^n$ and logical (*indicator*) variables $\delta \in \{0,1\}$. Consider the statement $X \triangleq [f(x) \leq 0]$, where $f : \mathbb{R}^n \Rightarrow \mathbb{R}$ is linear, assume that $x \in \mathcal{X}$, where $\mathcal{X}$ is a given bounded set, and define

$$M \triangleq \max_{x \in \mathcal{X}} f(x) \tag{B.2a}$$

$$m \triangleq \min_{x \in \mathcal{X}} f(x) \tag{B.2b}$$

M and m are over and under estimates, respectively. Therefore equation $y = \delta f(x)$ is equivalent to

$$\begin{aligned} &y \leq M\delta \\ &y \geq m\delta \\ &y \leq f(x) - m(1-\delta) \\ &y \geq f(x) - M(1-\delta) \end{aligned} \tag{B.3}$$

B.1.2 Mixed Logical Dynamical Systems

The tools obtained in the previous section will be used now to express relations describing the evolution of systems where physical laws, logic rules, and operating constraints are interdependent. We are able to do this by describing mixed logical dynamical (MLD) systems through the following linear relations [4]:

$$\begin{aligned} x(t+1) &= A_t x(t) + B_{1t} u(t) + B_{2t} \delta(t) + B_{3t} z(t) \\ y(t) &= C_t x(t) + D_{1t} u(t) + D_{2t} \delta(t) + D_{3t} z(t) \\ E_{1t} u(t) &+ E_{2t} \delta(t) + E_{3t} z(t) + E_{4t} x(t) \leq E_{5t} \end{aligned} \tag{B.4}$$

where $t \in \mathbb{Z}$,

$$x = \begin{bmatrix} x_c \\ x_l \end{bmatrix}, x_c \in \mathbb{R}^{n_c}, x_l \in \{0,1\}^{n_l}, n \triangleq n_c + n_l$$

is the state of the system, whose components are distinguished between continuous x_c and 0-1 x_l;

$$y = \begin{bmatrix} y_c \\ y_l \end{bmatrix}, y_c \in \mathbb{R}^{p_c}, y_l \in \{0,1\}^{p_l}, p \triangleq p_c + p_l$$

is the output vector,

$$u = \begin{bmatrix} u_c \\ u_l \end{bmatrix}, u_c \in \mathbb{R}^{m_c}, u_l \in \{0,1\}^{m_l}, m \triangleq m_c + m_l$$

is the command input, collecting both continuous commands u_c, and binary (on/off) commands u_l; $\delta \in \{0,1\}^{r_l}$ and $z \in \mathbb{R}^{r_c}$ represent respectively auxiliary logical and continuous variables. The continuous variable is defined as follows $z(t) = \delta(t)x(t)$.

B.2 Signal Temporal Logic (STL) constraints

Given a formula φ, we introduce a variable z_t^{φ}, whose value is tied to a set of mixed integer linear (MIL) constraints represented above. In other words, z_t^{φ} has an associated set of MILP constraints such that z_t^{φ} = 1 if and only if φ holds at position t. For z_0^{φ} represents the formula φ in the initial state and whether or not it holds.

The predicates are represented by constraints on system state variables. For each predicate $\mu \in P$, a binary variable $z_t^{\mu} \in \{0, 1\}$ is introduced. The following constraints enforce that z_t^{μ} = 1 if and only if $\mu(x_t) > 0$:

$$\begin{aligned} \mu(x_t) &\leq M_t z_t^{\mu} - \in_t \\ -\mu(x_t) &\leq M_t(1 - z_t^{\mu}) - \in_t \end{aligned} \tag{B.5}$$

where M_t is a sufficiently large positive number, and $\in_t$ a sufficiently small positive number that serve to bound $\mu(x_t)$ away from 0. M_t is just like M and m from section B.1 instead that M = M_t and m = $-M_t$.

The boolean operations on MILP variables described above are defined by logical operations. The logical operations on variables $z_t^{\varphi} \in [0, 1]$ are the following:

Negation: $z_t^{\varphi} = \neg z_t^{\psi}$ which can also be represented as $z_t^{\varphi} = 1 - z_t^{\psi}$, $z_t^{\varphi} \leq z_{t_i}^{\psi_i}, i = 1, \ldots, m$

Conjunction: $z_t^{\varphi} = \wedge_{i=1}^{m} z_{t_i}^{\psi_i}$ which can be represented as $z_t^{\varphi} \geq 1 - m + \displaystyle\sum_{i=1}^{m} z_{t_i}^{\psi_i}$, $z_t^{\varphi} \geq z_{t_i}^{\psi_i}, i = 1, \ldots, m$

Disjunction: $z_t^{\varphi} = \vee_{i=1}^{m} z_{t_i}^{\psi_i}$ which can be represented as $z_t^{\varphi} \leq \displaystyle\sum_{i=1}^{m} z_{t_i}^{\psi_i}$

Temporal operators $\Box$ and $\Diamond$ also get appropriate encoding:

Always: $\varphi = \Box_{[a,b]}\psi$ which can also be represented as $z_t^{\varphi} = \wedge_{i=a_t^N}^{b_t^N} z_i^{\psi}$

Let $a_t^N = \min(t + a, N)$ and $b_t^N = \min(t + b, N)$ where N is the receding horizon.

Eventually: $\varphi = \Diamond_{[a,b]}\psi$ which can also be represented as $z_t^{\varphi} = \vee_{i=a_t^N}^{b_t^N} z_i^{\psi}$

Until: $\varphi = \psi_1 \mathcal{U}_{[a,b]} \psi_2$

The definition of until is not relevant for this thesis so we don't represent in succinct detailed. However, its definition can be found in [28].

The combination of the STL constraints, system constraints and loop constraints gives the MILP encoding, and this enables checking feasibility of this set and find a solution using an MILP solver. After determining an objective function of the system, it is possible to find the optimal trajectory that satisfies the STL specifications.

B.3 Robustness-base encoding

In this section we give a short representation of the robustness function discussed in the section 2.4.3. The robustness can be computed recursively on the structure of the formula in conjunction with the generation of the constraints. In addition, since $\max$ and $\min$ operations can be expressed in an MILP formulation using additional binary variables, this does not add complexity to the encoding.

The encodings of the temporal operators work upon the encodings defined above. For each predicate $\mu \in P$, it is now introduced variables r_t^μ for time indices t = 0, 1, $\ldots, N$, and set r_t^μ = $\mu(x_t)$. For r_t^φ where φ is a Boolean formula, it is assumed that each operand φ has a corresponding variable $r_t^\varphi = \rho^\varphi(\mathsf{x}, t)$. The Boolean operations are defined as

Negation: $r_t^\varphi = \neg r_t^\psi$ which can also be represented as $r_t^\varphi = -r_t^\psi$

Conjunction: $r_t^\varphi = \wedge_{i=1}^m r_{t_i}^{\psi_i}$

$$\sum_{i=1}^m \rho_{t_i}^{\psi_i} = 1$$

$$r_t^\varphi \leq z_{t_i}^{\psi_i}, i = 1, \ldots, m$$

$$r^{\psi_i} t_i - (1 - \rho_{t_i}^{\psi_i})M \leq r_t^\varphi \leq r^{\psi_i} t_i + M(1 - \rho_{t_i}^{\psi_i})$$

Together, these constraints enforce that $r_t^\varphi = \min(r_{t_i}^{\psi_i})$.

Disjunction: $r_t^\varphi = \wedge_{i=1}^m r_{t_i}^{\psi_i}$

$$\sum_{i=1}^m \rho_{t_i}^{\psi_i} = 1$$

$$r_t^\varphi \geq z_{t_i}^{\psi_i}, i = 1, \ldots, m$$

$$r^{\psi_i} t_i - (1 - \rho_{t_i}^{\psi_i})M \leq r_t^\varphi \leq r^{\psi_i} t_i + M(1 - \rho_{t_i}^{\psi_i})$$

Together, these constraints enforce that $r_t^\varphi = \max(r_{t_i}^{\psi_i})$. The encoding for bounded temporal operators is defined as in section 2.4.3; The advantage of this encoding is that it allows to maximize robustness of satisfaction. Additionally, an encoding based on robustness has the advantage of allowing the STL constraints to be softened or hardened as necessary. However, due to additional binary variables is more computational expensive. On the other hand, the robustness constraints are more easily relaxed, which allows a simpler cost function to solve the same problem if it didn't had robustness in the objective function.